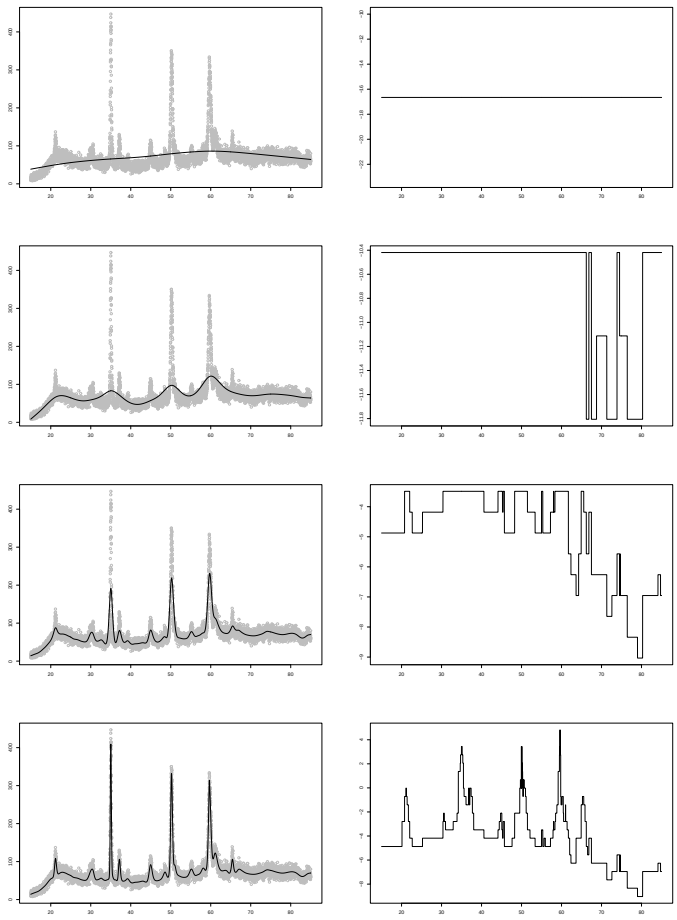
Residual Based Selection of Smoothing Parameters



Dissertation

zur Erlangung des akademischen Grades eines
Doktors der Naturwissenschaften
(Dr. rer. nat.)

dem Fachbereich Mathematik der Universität Duisburg-Essen vorgelegt im Juli 2004

von

Monika Meise, geb. in Düsseldorf

Tag der mündlichen Prüfung:	11. Oktober 2004
Gutachter:	Prof. Dr. P. L. Davies (Universität Duisburg-Essen)
	Prof. R. Koenker (University of Illinois)

For Adrian

Monika Meise
University of Duisburg-Essen
D-45117 Essen
Germany
monika.meise@uni-essen.de

Contents

Introduction

In many different situations people depend on the analysis of very special data sets. One example is given by the thin-film data set shown in Figure 1. Measuring the intensity of a reflected X-ray depending on its angle of incidence, one obtains information about the grid structure of a very thin film. In this context physicists are interested in the width and

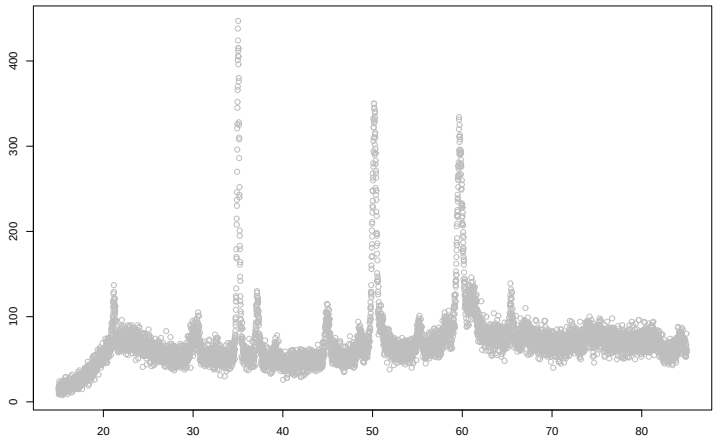


Figure 1: Thin-film data.

height of the occurring peaks. Hence a procedure is needed which simplifies the noisy data and moreover identifies the peaks. This requirement is equivalent to the problem of finding a separation between the trend of the baseline and the more important peaks.
This example illustrates the problem of analyzing data sets with heterogeneous variability. In addition to uniform parts very thin or small but important features occur. Procedures for a satisfactory data analysis should be able to approximate both the smooth parts, that have only small variation, as well as the highly variable parts. In an optimal way this should be done completely automatically, without any additional information. In many cases such a procedure can depend on special knowledge about the topic and information about how the data were obtained. But nevertheless often standard methods are used for a first and very quick analysis. Therefore numerous variations of the standard nonparametric regression methods have been developed, as there are kernel estimators, local polynomials and

smoothing splines. But most of them, as we think, are not able to provide an approximation in cases like the thin-film data which satisfy the different aspects, mentioned above.
Since all of these standard methods depend on the determination of at least one smoothing parameter our goal was to find an automatic procedure for its specification, which also gives satisfactory results for such heterogeneous and hence very complicated data sets. Obviously the residuals of an approximation are able to tell us a lot about its quality. One procedure which exploits these properties very efficiently is the taut string method of Davies and Kovac (2001). Here the so-called multiresolution conditions give a mathematical description how the residuals can be used to control the approximation of a data set.
In the present work we show how this residual analysis can be combined with established nonparametric regression methods. We obtain an automatic procedure to determine the needed smoothing parameters and achieve results with improved local flexibility for each of the three methods, mentioned above. Considering two different real data sets we show how these new procedures can be modified to cope with the particular problems of the data. To separate peaks and the baseline from the thin-film data we combine the weighted smoothing splines with the taut string approximation. A second data set, the balloon data, contains outliers from the way how it has been measured. Hence we provide a robust version of our approximation procedure.
The second part of this work is concerned with different procedures for analyzing two dimensional data. First we propose a robust procedure for bivariate data which contain very large outliers. The approximation for such data is given by the minimizer of an L_1-fidelity term penalized by a term based on the total variation norm. The locally defined smoothing parameters are chosen automatically using a two dimensional version of the multiresolution analysis of the residuals. Additionally we show how these multiresolution conditions can be used for a satisfactory selection of the smoothing parameter for thin plate splines and penalized triograms.
For the computation of the examples we used our own source codes in C and the statistics software R with some additional functions of available packages. The basic parts of the programs for the univariate procedures can be found in the Appendix A.

I want to thank professor P. L. Davies for introducing me to this interesting area of research and for many stimulating discussions on the subject matter of this thesis. Moreover I am grateful to him for all his help which made it possible that I could stay two months at the University of Illinois in Urbana-Champaign and I thank the DAAD for financial support. Also I want to thank professor R. Koenker for many interesting and helpful suggestions during the time in Illinois and afterwards. I appreciate the constant help of Dr. A. Kovac and his most valueable motivating support. Finally I want to thank my father for not becoming tired reading and discussing the finished parts of this work during recreative Sundays with delicious meals (thank you, mom!).

Chapter 1

Linear Smoothing Procedures

Following the basic idea that we outlined in the introduction, it is reasonable to assume that the data (t_i, y_i) with $i = 0, \ldots, n$ rely on a signal plus some noise. This case occurs very often. Thus in many circumstances the consideration of the following model seems to be helpful:

$$y_i = f(t_i) + \varepsilon_i \qquad i = 0, \ldots, n. \tag{1.1}$$

Here $f(t_i)$ is an idealized description of the signal and the ε_i are considered as noise.
Satisfactory methods of data analysis in most cases need to depend on special properties of the data, but nevertheless also general methods can help to interpret the data. In this context one is interested in simple methods, which provide a better visualization of the data by reducing the complexity and concentrating the information to its important part.
In the following we want to give some examples of such very common so called *smoothing methods*, as there are kernel estimators, local polynomials and splines.
Each of them belongs to the family of linear smoothers, which means that the resulting approximation $\hat{f}(t_i)$ can be written as a linear combination of the observations y_i

$$\hat{f}(t_i) = \sum_{j=0}^{n} S_{i,j} y_j. \tag{1.2}$$

with the weighting or smoothing matrix $\mathbf{S}$ depending on the properties of the method.
In the sequel a very short overview will be given about basic ideas and computational aspects of these different procedures.

1.1 Nearest Neighbor

Visualizing data without any noise reduction often does not seem to be very informative. This is the reason for the existence of so many data smoothing techniques. The most commonly used method is local averaging. For this procedure the data are approximated locally by one constant, the average of the data values in a specified (time) window. The

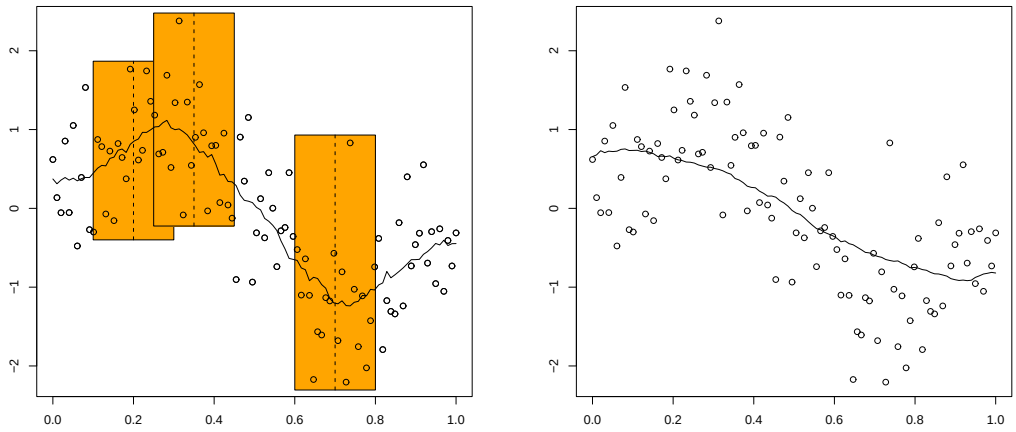


Figure 1.1: Two moving average results of a noisy sine curve (left: $h = .1$, right: $h = .3$). The orange regions display the considered data window for the local approximation.

so called bandwidth h determines the size of the considered interval $I_j = \{i : |t_j - t_i| \leq h\}$. The smoothed value $\hat{f}$ at the design point t_j then is given by

$$\hat{f}(t_j) = \frac{1}{N_j} \sum_{i \in I_j} y_i, \tag{1.3}$$

with $N_j = \#I_j$. The window can be regarded as moving along the time scale with the center always situated in the next data point t_i (cf. Figure 1.1). This simplest version of a moving average estimator always considers data values in the neighborhood of the location t_i and thus belongs to the nearest neighbor estimators.
Figure 1.1 shows the function $f(x) = \sin(2\pi x)$ with added noise. The smoothed curve on the left-hand side is the result of the moving average with bandwidth $h = 0.1$, the second approximation uses $h = 0.3$. Obviously a larger bandwidth causes a smoother approximation whereas local means using a smaller bandwidth are able to reproduce more details of the original data set. Both local mean approximations are not very smooth. It will be shown that improved nearest neighborhood procedures obtain better smoothing results.

1.1.1 Kernel estimators

Procedures which provide an improved local averaging are the kernel regression estimators. By introducing local weights it is possible to compute local means where the influence of the data points depend on the closeness to the location of the current design point.

Roughly speaking this means that the smaller $|t_j - t_i|$ the larger is the weight for y_i in the computation of $\hat{f}(t_j)$. For the weighting usually second order kernel functions are used.

Definition 1.1. *A measurable, positive function $K : \mathbb{R} \to \mathbb{R}$ is called a kernel function of order k, if*

$$\int K(u)du = 1, \qquad \int u^j K(u)du = 0 \quad j = 1, \ldots, k-1, \quad \textit{and} \quad \int u^k K(u)du = \alpha \neq 0.$$

The kernel functions are not required to have compact support but the following table of familiar kernel functions shows that most of them provide weights for the window $[-1, 1]$.

kernel functions	$K(x)$
uniform	$\frac{1}{2}\mathbf{1}_{\{\lvert x\rvert \leq 1\}}$
triangle	$(1 - \lvert x\rvert)\mathbf{1}_{\{\lvert x\rvert \leq 1\}}$
epanechnikov	$\frac{3}{4}(1 - x^2)\mathbf{1}_{\{\lvert x\rvert \leq 1\}}$
quartic	$\frac{15}{16}(1 - x^2)^2\mathbf{1}_{\{\lvert x\rvert \leq 1\}}$
triweight	$\frac{35}{32}(1 - x^2)^3\mathbf{1}_{\{\lvert x\rvert \leq 1\}}$
gaussian	$\frac{1}{\sqrt{2\pi}} \exp(-\frac{1}{2}x^2)$
cosinus	$\frac{\pi}{4} \cos(\frac{\pi}{2}x)\mathbf{1}_{\{\lvert x\rvert \leq 1\}}$

These kernel functions are for example used in one of the oldest and best known kernel regression methods, the local weighting of the Nadaraya-Watson estimator (Nadaraya, 1964; Watson, 1964)

$$\hat{f}_h(t_j) = \sum_{i=0}^{n} \frac{K\left(\frac{t_j - t_i}{h}\right)}{\sum_{l=0}^{n} K\left(\frac{t_j - t_l}{h}\right)} y_i \,. \tag{1.4}$$

For the uniform kernel, the Nadaraya-Watson estimator becomes the moving average estimator (1.3).
The choice of an appropriate kernel function is not relevant for the further work. Remarks on this and other topics in the context of kernel estimators can be found for example in Eubank (1988, 1999), Härdle (1990) and Wand and Jones (1995). Alternative kernel regression estimators were introduced by Priestley and Chao (1972) and Gasser and Müller (1979,1984). Comparisons can be found for example in Chu and Marron (1991). Gasser and Müller (1979) and also Müller (1991) discuss the possibilities of special boundary kernels. The reflections about the effective kernel idea in Wand and Jones (1995) are based on Hastie and Loader (1993), Silverman (1984) and Hastie and Tibshirani (1990).

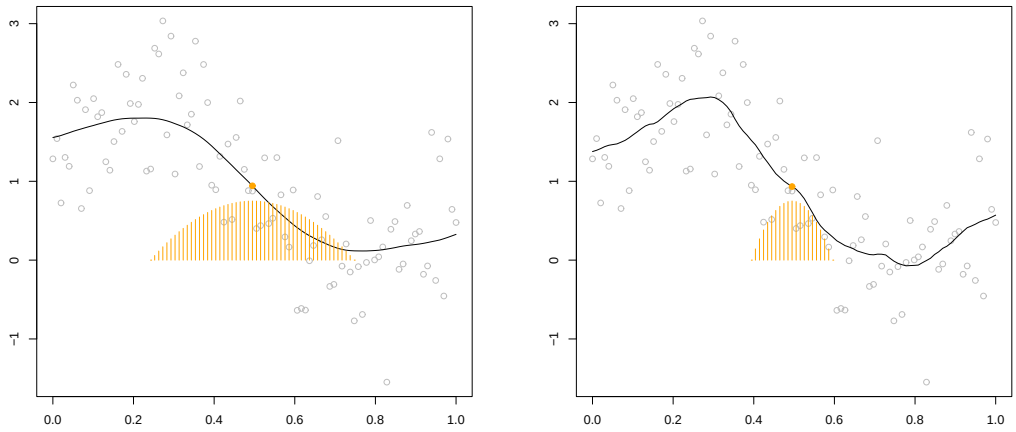


Figure 1.2: Approximation of a noisy sine curve using the NWE with Epanechnikov kernel function. Left: $h = 0.25$, right: $h = 0.1$. The orange hatched area shows the influence of the corresponding data points for the approximated value at the marked point.

Figure 1.2 shows the difference between two approximations of a noisy sine curve using the Nadaraya-Watson kernel estimator with different bandwidths. The relatively large bandwidth $h = 0.25$ in this case (number of data points is $n + 1 = 100$) means that each local adaptation is computed by the weighted mean of 51 points. This leads to over-smoothing of the data. The two extreme values are not as distinct as one would expect for a satisfactory approximation. In contrast to this the figure on the right-hand side shows the approximation for $h = 0.1$. Now some features occur in the fitted curve which cannot be found in a usual sine curve, namely some suspicious extreme values, so called artefacts. As mentioned before, a smaller h improves the data closeness. For equispaced data (t_i, y_i), $i = 0, \ldots, n$ on $[0, 1]$ the result of a Nadaraya Watson approximation using $h < \frac{1}{n}$ is just the interpolation of the data.
Obviously, the main question in connection with kernel estimators is how to find an appropriate bandwidth.

Bandwidth Selection

Typically the bandwidth is chosen *asymptotical optimal* in the sense that the resulting approximation minimizes a measure of goodness of fit under special smoothness assumptions for f. In the kernel regression case usually this is the mean square error (MSE)

$$MSE(h) = \frac{1}{n+1} \sum_{i=0}^{n} \left(f(t_i) - \hat{f}_h(t_i))^2 dx \right),$$

or the integrated mean squared error (MISE). From the numberless papers about this topic we want to mention as references just Müller and Stadtmüller (1987) and Staniswalis

(1989).

One important technique for asymptotical optimal bandwidth selection is provided by plug-in bandwidth selectors, which rely on an estimation of the functional $\|f''\|_2^2$, where $\|g\|_2^2 = \int_{-\infty}^{+\infty} g(u)^2 du$ is the L_2-norm.
Another commonly used procedure selects the bandwidth via minimizing some cross validation criterion

$$CV(h) = \frac{1}{n+1} \sum_{i=0}^{n} (y_i - \hat{f}_{h,i}(t_i))^2 \tag{1.5}$$

where $\hat{f}_{h,i}$ is the estimator without using the i-th observation.
Publications which concentrate on the CV technique and propose different bandwidth selectors are for example Härdle and Marron (1985), Härdle, Hall and Marron (1988), Gasser, Kneip and Köhler (1991) and Härdle, Hall and Marron (1992).
The idea to chose the bandwidth optimal in respect to some measure of goodness of fit results from the consideration that a satisfactory approximation of the data has to be in some sense close to the data. This closeness is commonly measured in the L_2-norm which will be discussed at some later point but leads to other regression methods.

1.1.2 Local Polynomial Regression

Data closeness is essential for a reasonable approximation. Therefore smoothing methods often rely on the idea to minimize the distance between the data and the approximation requiring special properties of the approximating function $\hat{f}$. Measuring the distance in the L_2-norm leads to least squares minimization problems. Requiring f to be constant, the simple local kernel approximation $\hat{f}_h(t_j)$ in (1.4) can be obtained as the result of the following least squares minimization problem:

$$\min_f \sum_{i=0}^{n} (y_i - f(t_i))^2 \, K\left(\frac{t_j - t_i}{h}\right). \tag{1.6}$$

This equivalence becomes obvious by solving the minimization problem using (1.8) for this special case. It is reasonable to generalize the locally constant approximation which is provided by kernel estimators to linear functions or polynomials of higher order.
If $\hat{f}_h$ is assumed to be $(p+1)$-times differentiable it is possible to approximate it by a Taylor expansion of order p as

$$\hat{f}(t) \approx \hat{f}(t_j) + \hat{f}'(t_j)(t - t_j) + \frac{\hat{f}''(t_j)}{2!}(t - t_j)^2 + \cdots + \frac{\hat{f}^{(p)}(t_j)}{p!}(t - t_j)^p.$$

Adapting this to the least squares estimator (1.6), we obtain

$$F(\mathbf{b}) = \sum_{i=0}^{n} \left(y_i - \sum_{l=0}^{p} b_l (t_i - t_j)^l \right)^2 K\left(\frac{t_i - t_j}{h}\right) \tag{1.7}$$

with $\mathbf{b} = (b_0, \ldots, b_p)$. That means $\hat{f}_h(t_j) = \hat{b}_0$ with $\hat{\mathbf{b}} = \arg\min_{\mathbf{b}} F(\mathbf{b})$.

Following the least squares theory, the approximation at the design point t_j can be computed easily by using the design matrix

$$\mathbf{X} = \begin{pmatrix} 1 & (t_0 - t_j) & \cdots & (t_0 - t_j)^p \\ \vdots & \vdots & & \vdots \\ 1 & (t_n - t_j) & \cdots & (t_n - t_j)^p \end{pmatrix}$$

and a weighting matrix

$$\mathbf{W} = \operatorname{diag}\left\{ K\left(\frac{t_i - t_j}{h}\right)\right\}.$$

With

$$\hat{\mathbf{b}} = \begin{pmatrix} \hat{b}_0 \\ \vdots \\ \hat{b}_p \end{pmatrix} \qquad \text{and} \qquad \mathbf{y} = \begin{pmatrix} y_0 \\ \vdots \\ y_n \end{pmatrix}$$

the least squares problem can be written as

$$\hat{\mathbf{b}} = \min_{\mathbf{b}} (\mathbf{y} - \mathbf{X}\mathbf{b})^T \mathbf{W} (\mathbf{y} - \mathbf{X}\mathbf{b}),$$

$\mathbf{b} = (b_0, \ldots, b_p)^T$. Then the solution is given by

$$\hat{\mathbf{b}} = (\mathbf{X}^T \mathbf{W} \mathbf{X})^{-1} \mathbf{X}^T \mathbf{W} \mathbf{y}. \tag{1.8}$$

Good overviews about local polynomial regression are given for example in Fan and Gijbels (1996) and Ruppert and Wand (1994). Their remarks trace back to Stone (1977), Cleveland (1979), Fan (1992, 1993) and Fan and Gijbels (1992)

It is easy to see, that the local polynomial estimator can be written in the sense of linear estimators (1.2) with:

$$S_{i,j} = e_1^T (\mathbf{X}^T \mathbf{W} \mathbf{X})^{-1} \{1, t_i - t_j, \ldots, (t_i - t_j)^p\}^T K\left(\frac{t_i - t_j}{h}\right).$$

Bandwidth and Order Selection

Each of the bandwidth selection techniques mentioned above can also be adapted to the local polynomial approximation. Beside the bandwidth also the choice of the order p of the locally adapted polynomials is important for the regression results. Figure 1.3 shows the difference between two local polynomial approximations using the same bandwidth but with varying polynomial degree. On the left hand side a local linear approximation, on the right hand side a cubic one. Obviously the cubic approximation obtains a better

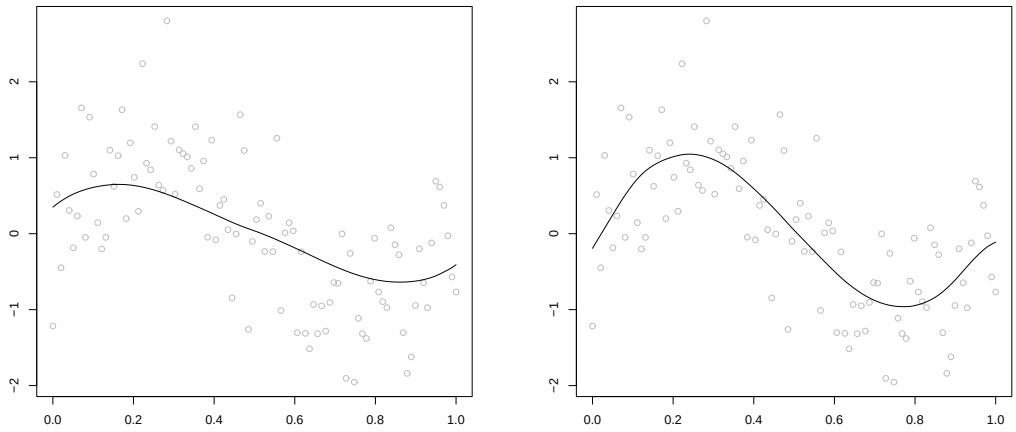


Figure 1.3: Local polynomial approximation of a noisy sine curve with $h = 0.5$ (left: degree of the polynomial $p = 1$, right: $p = 3$)

development of the two extreme values. With the local linear version such a good approximation of the extreme values would only be possible using a smaller bandwidth. Hence the approximation would not be as smooth as in the cubic case.

Fan and Gijbels (1996) explain in respect to the problem how to choose p the phenomenon of the *odd world*. As it is also described in Fan (1992), Fan and Gijbels (1992), Hastie and Loader (1993) or Ruppert and Wand (1994) the change from an even to its consecutive odd order does not increase the variance. The additional parameter is an opportunity for a significant bias reduction. Moreover they point out that even order fits suffer from low efficiency. Another possibility is to choose the degree p variable. Again it is proposed to determine p by minimizing the estimated local mean squared error (MSE), now with respect to p. In practice it is necessary to find a compromise between the advantage of higher order polynomials and the increasing computational effort. Therefore $p = 3$ is a common choice when local polynomials are used.

1.2 Roughness Penalty

Local polynomial regression combines the least squares approximation with the restriction that the locally adapted function is a polynomial. But the idea of the least squares minimization also leads to another regression approach. Obviously $\hat{f}$ is the closer to the data the smaller $F(\hat{f}) = \sum_{i=0}^{n}(y_i - \hat{f}(t_i))^2$ is. Without any additional conditions the minimization of F leads to undesirable interpolation of the data. Therefore one combines the fidelity term F with a penalty term P, which penalizes certain properties of the adapted function.

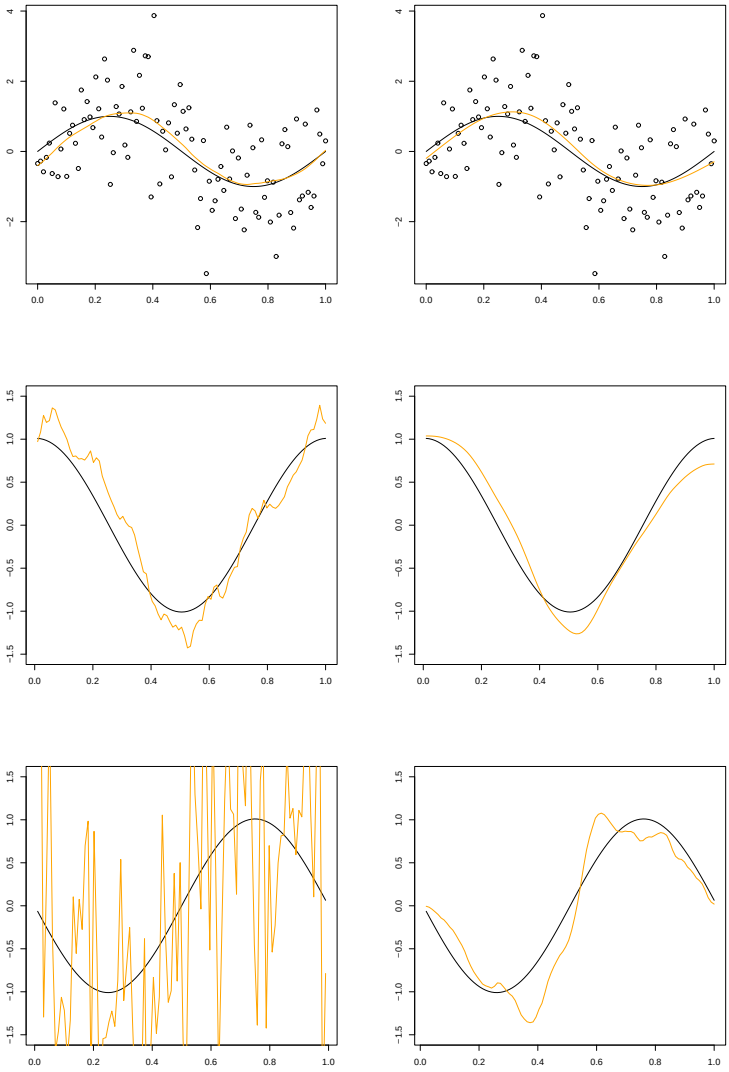


Figure 1.4: A local polynomial and a spline approximation of a noisy sine curve (top) and their first (middle) and second (bottom) order differences, in black the original function and their derivatives.

1.2.1 Smoothing Splines

In the case of smoothing splines a criterion to measure the roughness of the adapted function, the norm of the second derivative, is used to penalize the least squares problem. Then the regression function would be $\hat{f}_\lambda$ twice differentiable so that $\hat{f}_\lambda$ is minimizer of $S_\lambda(f)$ with

$$S_\lambda(f) = \sum_{i=0}^{n} (y_i - f(t_i))^2 + \lambda \int_a^b (f''(x))^2 \, dx \tag{1.9}$$

Definition 1.2. *$t_0, \ldots, t_n$ be real values in $[a, b]$, more precisely $a < t_0 < \cdots < t_n < b$. A function g defined on $[a, b]$ is a* cubic spline *with* knots *t_i, $i = 0, \ldots, n$, if*

(a) g is a cubic polynomial on $(a, t_0), (t_0, t_1), \ldots, (t_n, b)$

(b) g is twice continuously differentiable on $[a, b]$.

A cubic spline with vanishing second and third derivative in a and b is called a natural cubic spline*.*

For $\hat{f}_\lambda$ minimizing $S_\lambda(f)$ on the Sobolev space $W_2^2[a, b]$ it can be shown that $\hat{f}_\lambda$ is a natural cubic spline. $\hat{f}_\lambda$ exists and is unique. The knowledge of these properties of $\hat{f}_\lambda$ is the basis of many fast and efficient algorithms that calculate an approximating spline. One of these algorithms is the *Reinsch algorithm* (Reinsch, 1967). Its basic idea is to employ a non-singular system of linear equations of the second derivative in the knot points t_i. A favorable characteristic of this system is its banded structure which reduces memory requirements and allows a fast Cholesky decomposition.

For an introduction into the properties and possibilities of smoothing splines see Wahba (1990), Green and Silverman (1994) and Eubank (1988, 1999). The roughness penalty approach can also be found in Hastie and Tibshirani (1990) and Rosenblatt (1991). Fundamentals about Splines are summarized in de Boor (1978, 2001) and Schumaker (1981). They also present an extensive bibliography about splines on the Internet.

Because of their differentiability splines are of special interest for regression problems which rely on the smoothness of the approximation. Figure 1.4 for example shows two similar approximations, on the left hand side (first row) the result of a local linear approximation using bandwidth $h = 0.2$ on the right hand side the result of the smoothing spline with $\lambda = 0.8$. But the difference between these two approximations becomes obvious in their first and especially their second order differences. The first order differences both seem to be relatively close, in the sense of residual sums, to the original first derivative. But only the spline differences are able to approximate also the smoothness of the original derivative. Hence the second order differences provide a good approximation of the original second derivative, whereas the second order differences of the local polynomial approximation fail completely. This property of cubic smoothing splines can be very useful in special regression problems, cf. section 5.3.

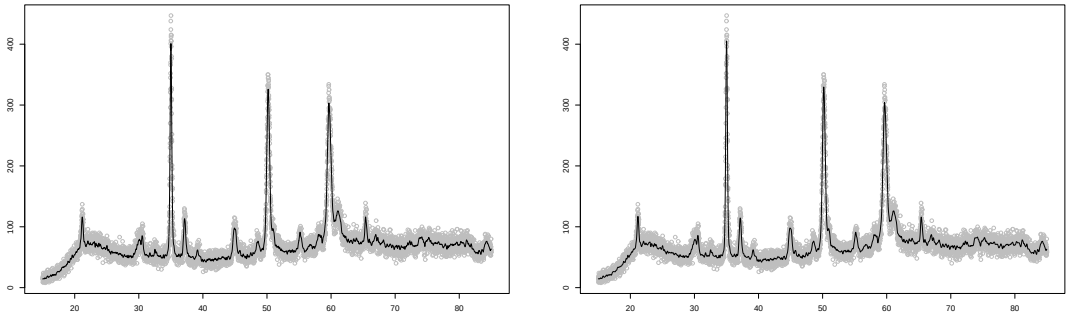


Figure 1.5: Left: kernel estimator approximation with plug-in bandwidth selection, right: smoothing spline approximation using GCV

Cross-Validation

Of course also in the case of smoothing spline approximation the main question for a satisfactory result is how to choose the smoothing parameter λ, which is needed to guaranty the balance between a good approximation and the amount of smoothness of the resulting function. For $\lambda \to 0$ the data are almost interpolated whereas for $\lambda \to \infty$ f converges to the linear least squares approximation.
As mentioned before, usually also the smoothing parameter for splines is chosen asymptotically optimal in respect to some goodness of fit criterion, as the MSE. Hence the cross validation method is computationally intensive, Wahba (1977) and Craven and Wahba (1979) proposed an improved version, the generalized cross-validation (GCV). This method uses the properties of the three described smoothing methods which rely on the fact that they all belong to the family of linear estimators. Also the smoothing splines can be written in respect to 1.2. For example Cox (1983) and Silverman (1984, 1985) point out the connections between kernel estimators and smoothing splines. Silverman (1984) shows that the smoothing spline can be asymptotically written as a local kernel average using the so called effective kernel function. For large n and local density $g(t_i)$ at the design points t_i, the resulting weights in (1.2) then are

$$S_{i,j} = \frac{1}{g(t_i)} K_h(t_j - t_i)$$

with the effective kernel function[1]

$$K(x) = \frac{1}{2} \exp\left(-\frac{|x|}{\sqrt{2}}\right) \sin\left(\frac{|x|}{\sqrt{2}} + \frac{\pi}{4}\right). \tag{1.10}$$

The GCV now uses the fact that the fitted values $\hat{f}$ can be expressed as $\hat{f} = \mathbf{S}(\lambda)\,\mathbf{y}$ and

[1] h in this case is a locally defined parameter which satisfies $h(t) = \lambda^{\frac{1}{4}} n^{-\frac{1}{4}} g(t)^{-\frac{1}{4}}$.

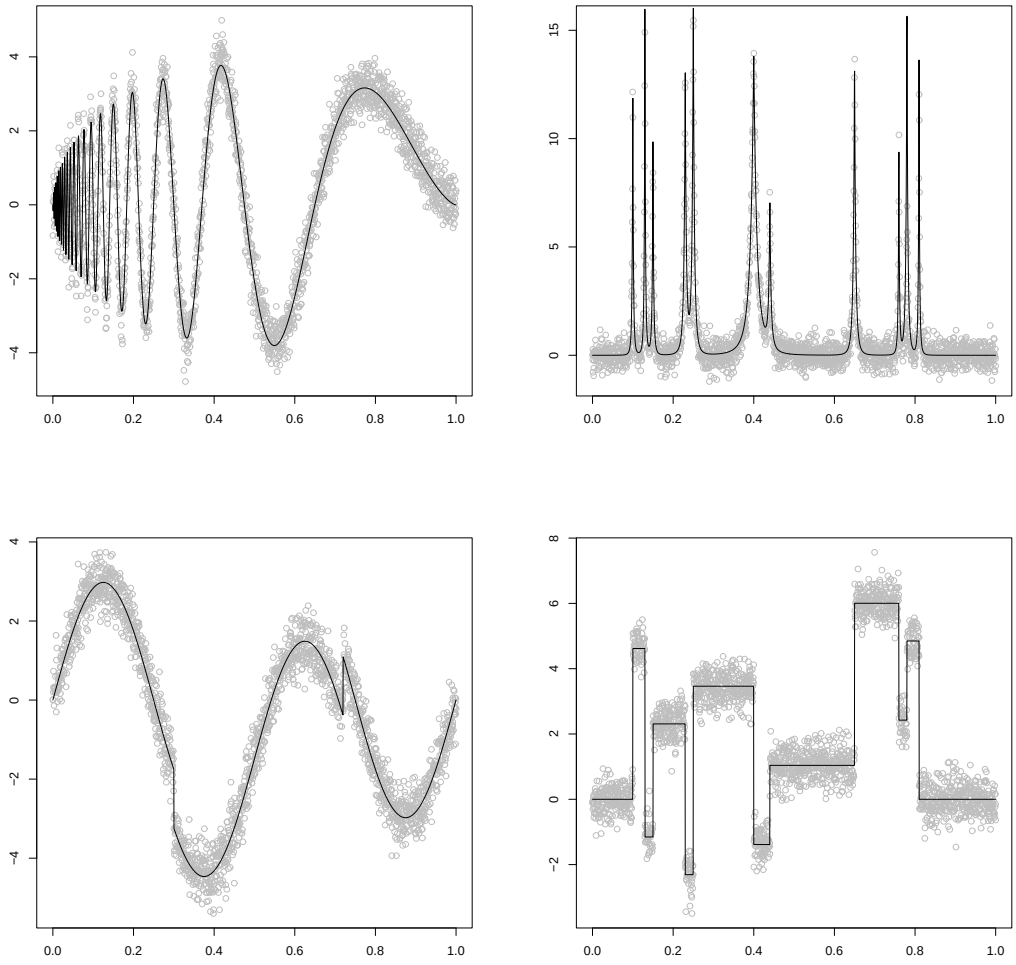


Figure 1.6: The four Donoho and Johnstone data sets (grey points), and the original functions (black line), top left: Doppler, top right: Bumps, bottom left: Heavisine, bottom right: Blocks.

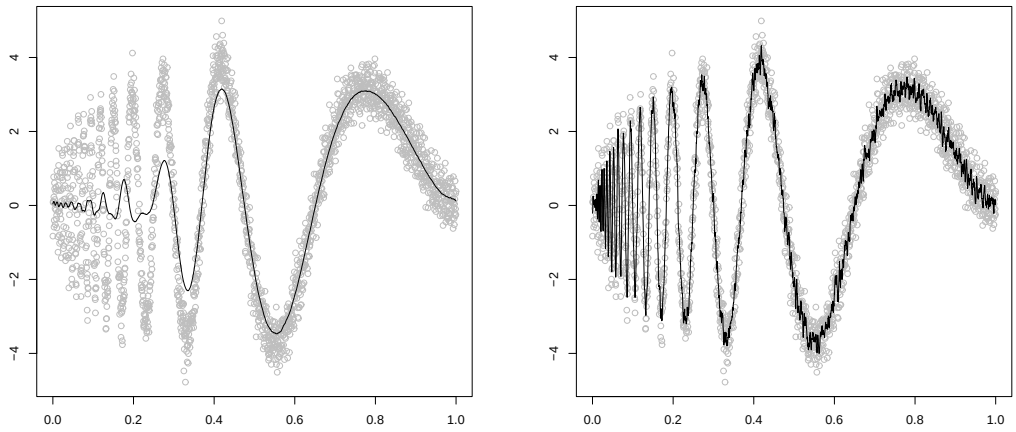


Figure 1.7: Kernel approximation of the Doppler data, left: $h = 0.07$, right: $h = 0.02$.

selects the smoothing parameter so that it minimizes

$$GCV(\lambda) = n^{-1} \frac{\sum_{i=0}^{n} (y_i - \hat{f}(t_i))^2}{1 - n^{-1} \operatorname{tr} \mathbf{S}(\lambda)}. \tag{1.11}$$

1.2.2 Variable Smoothing Parameter

As long as the data sets are relatively homogeneous and do not contain smoother parts and at the same time intervals with heavy fluctuations the described procedures using the plug-in or cross-validation based smoothing parameter selection work very well, but what happens in cases as for example the thin-film data? As described before, physicists are interested in the trend of the baseline and the location, width, and height of the peaks. These requirements show that a method is needed which is able to recover the smooth parts of the data and at the same time is flexible enough to adapt the peaks. Figure 1.5 shows (left) the approximations of the kernel estimator with plug-in bandwidth selector from Brockmann, Gasser and Herrmann (1993) and the result of a smoothing spline using GCV for the smoothing parameter selection (right). Obviously both methods obtain almost the same fit and none of them is able to approximate a smooth baseline. This is the reason why even though the approximation of the peaks is very good, both results can not be used to decide, which peaks are real and hence interesting peaks, and finally to separate them from the base line. The approximation of the base line parts is very rough and contains many artefacts, additional small peaks, which could not be distinguished from real peaks easily.

The two different but similar approximation results[2] indicate the main problem of the

[2]The difference in the resulting values is so marginal that it is hardly visible.

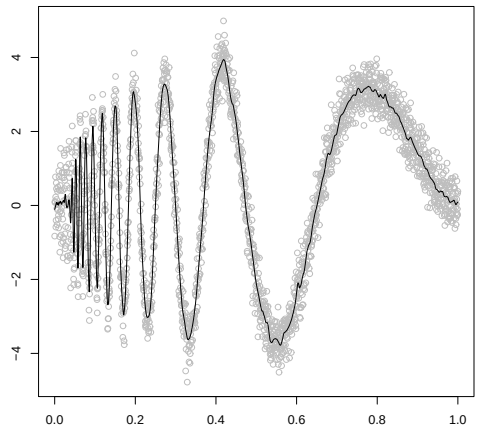


Figure 1.8: Doppler approximation with local plug-in bandwidth selection.

techniques described so far. Each of them relies on the selection of only one single (global) smoothing parameter, the bandwidth h or the penalty parameter λ.
In the sequel we want to use four artificial data sets which were proposed by Donoho and Johnstone (1994) to explain the problems and possibilities of data approximation. Each of them (cf. Figure 1.6) contains special difficulties.
Obtaining a satisfactory kernel approximation result for the Doppler data (Figure 1.6, top left) is obviously not possible. A relatively large bandwidth as e. g. $h = 0.07$ might be able to approximate the smooth right part of the data set, but of course has no chance to capture the fast oscillations in the left part (cf. Figure 1.7). In contrast to this the approximation using $h = 0.002$ adapts the left part of the data very good but is too wriggly in the right part.

The most obvious way to reach more local flexibility in the case of kernel estimators or local polynomials is to replace the global bandwidth h by a locally defined one $h = (h_0, \ldots, h_n)$. Hence the width of the considered data window, in the case of a kernel function K with finite support, is chosen so that it depends on the design point t_j. The local weights in (1.4) and (1.6) then become

$$K\left(\frac{t_j - t_i}{h_j}\right).$$

But still the problem remains how to choose these local bandwidth values. There are several publications about the local bandwidth selection problem. Common software works with a local version of the GCV or with plug-in procedures which have been adapted to the local case. One plug-in technique was implemented by Herrmann (1997). The result of this automatic local bandwidth selection method can be seen in Figure 1.8. It shows again that the main idea of minimizing an estimate of the L_2-error tends to stress the data closeness.

As a consequence, the approximation tends to have artefacts and is not rather smooth.
In the case of smoothing splines it is not as easy as in the context of kernel estimators to obtain a localized version. Replacing the global λ by a locally defined parameter would disable the usually used computational method. Instead of doing so, local weights are added in the fidelity term, so that a weighted, penalized least squares problem has to be minimized. We will introduce these weighted smoothing splines in chapter 4.

Chapter 2

Adequate Approximation

As mentioned earlier, the plug-in or cross-validation selection methods for smoothing parameters are optimal in the sense that they minimize the least squares error. This is the reason why also the local bandwidths tend to be relatively small and the resulting approximations are not always smooth. Before we propose our method for local bandwidth selection we want to motivate our procedure by specifying what we expect from a satisfactory data approximation. In other words we want to specify which properties we expect from an, as we say, *adequate approximation*.

2.1 Disambiguation and Goodness-of-Fit Criteria

Our requirements to an adequate approximation are closely related to the model (1.1). Since we expect the data to be the result of a signal plus noise, we want to use this knowledge for our method. Every smoothing technique tries to achieve a reasonable noise reduction of the data to make the interpretation of its visualization easier. Our approach is similar, we try to split the data into two parts. One part should finally contain the important information of the data and the other one the nonrelevant part. In our model we can address most of the complexity, the expected noise, to the first mentioned data component. This means that our approximation $\hat{f}(t_i)$, $i = 0, \ldots, n$ needs to adapt the data in a way that after subtracting it from the data y_i, $i = 0, \ldots, n$ everything which remains can be explained as noise and does not contain any important data information. On the other hand the approximation itself should contain all the important information but remain simple in contrast to the residuals.

2.1.1 Residuals

According to this basic idea, structure and location of the residuals $r_i = y_i - \hat{f}(t_i)$, $i = 0, \ldots, n$, finally have to correspond to the properties of the expected noise. Of course there is a variety of different kinds of noise, but to make the way of thinking clear we restrict our explanations to the case of white noise, thus the ε_i in (1.1) are i.i.d. $N(0, \sigma^2)$. Overall

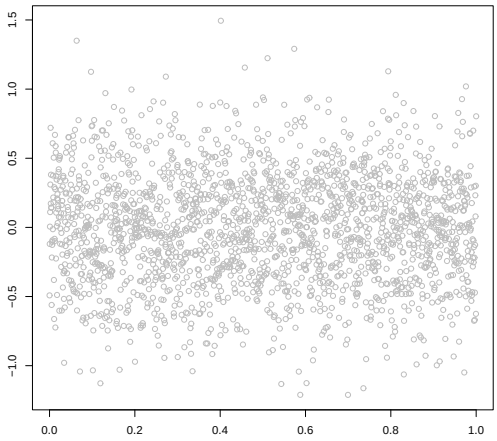


Figure 2.1: Simulated normally distributed data.

one could say we are looking for a data approximation so that the resulting residuals look like white noise. For other kinds of noise it would be possible or sometimes also necessary to find different specifications, as it will be shown in section 5.1.

One possibility to decide whether the residuals of an approximation are approximately normally distributed, would be a standard testing strategy. But our aim is to develop a method to determine the values of a locally defined bandwidth. Therefore we are not only interested in the fact that the residuals are approximately normally distributed but also in the question in which part they already look like white noise and in which not. Only this information can help to decide if a bandwidth locally needs to be reduced or not. For that reason a different decision strategy is needed.

Figure 2.2 (left) shows a similar approximation to the Doppler data as in Figure 1.7. Obviously, the resulting residuals (right) do not even approximately look like the simulated noise ($N(0, 0.4)$ in Figure 2.1). They still feature a clearly recognizable structure, which is an indication that they still contain important data information. This decision is easy for a personal observer, but to do it by a computer program a mathematical description is needed.
The approximation shows that almost all of the extreme values are over-smoothed, maxima are under- and minima over-estimated. If we compare this impression with the corresponding regions of the residuals, obviously their absolute values are too large compared to normally distributed data. But even more. The sums of the residuals on different intervals would also be able to distinguish between small regions where for example a maximum is hardly under-estimated or larger parts where the approximation only differs a little from a good fit. In both cases the absolute values of the sums of the residuals on the appropriate

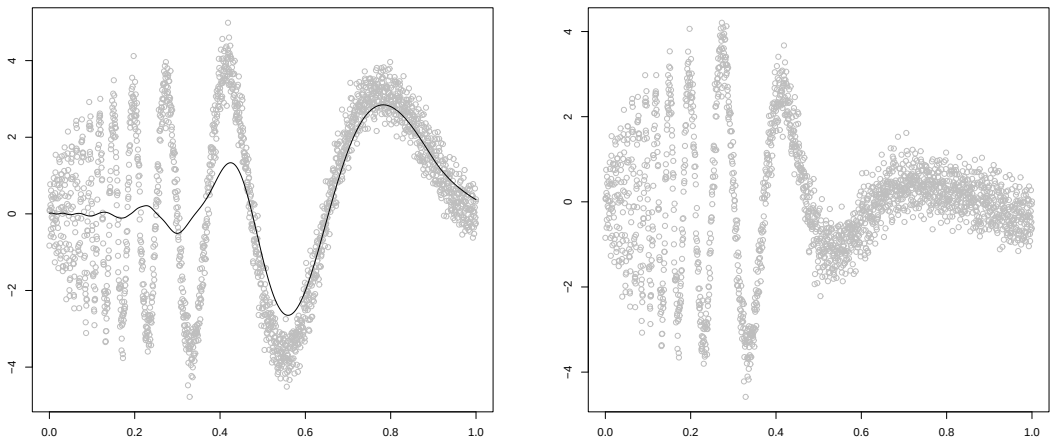


Figure 2.2: One result of a kernel estimator using bandwidth $h = 0.1$ for the Doppler data set (left) and its residuals (right).

intervals are larger than the ones of normally distributed data. Whereas on intervals with good fits these sums would correspond in their size to the ones of white noise.
In order to make this more precise one considers the sums of the residuals w_I on intervals I,

$$w_I = \frac{1}{\sqrt{N_I}} |\sum_{i \in I} r_i|. \tag{2.1}$$

In the case of white noise the size of these sums can be deduced from the characteristics of the normal distribution. Using its tail properties it is easy to show that for $Z_1, Z_2, \ldots$ i.i.d. $\mathbb{N}(0, \sigma^2)$

$$\lim_{n \to \infty} \mathbb{P}\left(\max_{1,\ldots,n} |Z_i| \leq \sigma \sqrt{\tau \log(n)} \right) = 1 \tag{2.2}$$

for some $\tau > 2$.
It is known that this holds also for sums of i.i.d random variables $Z = \frac{1}{\sqrt{n}} \sum_1^n Z_i$. The idea behind the criteria, which we are going to use, so-called multiresolution criteria, is to check whether these sums of the residuals r_i behave like $N(0, \sigma^2)$ i.i.d random variables or not. This is said to be the case if

$$w_I \leq \sigma \sqrt{\tau \log(n)} \tag{2.3}$$

for all intervals I.
For a detailed description of the multiresolution idea see e. g. Donoho (1995) and Davies and Kovac (2001).

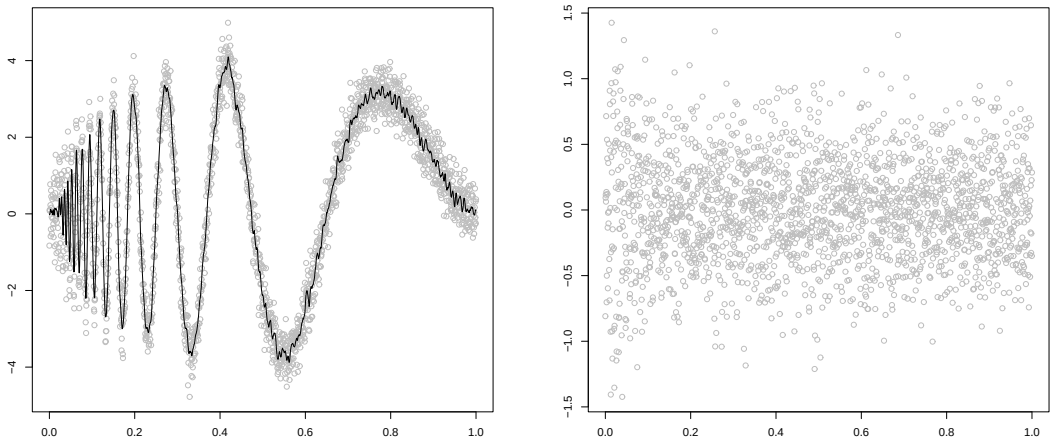


Figure 2.3: One result of a kernel estimator using bandwidth $h = 0.003$ for the Doppler data set (left) and its residuals (right).

2.1.2 Simplicity

In contrast to the over-smoothing kernel approximation Figure 2.3 shows the kernel estimation result for the Doppler data using $h = 0.003$. This approximation fulfills the requirements (2.3) and thus the residuals provide a better approach to normally distributed data. But now obviously the right part of the approximation is very wriggly. The adaptation possesses artefacts without any justification by the data. This demonstrates clearly that for an adequate approximation it is not enough to require only that the residuals satisfy the normal noise criterion. We also expect the signal to be simple. Hence the approximation needs to be smooth. In fact we are looking for a regression result which satisfies the residual criteria and is as smooth as possible at the same time.

Chapter 3

Multiresolution based Bandwidth Selection

3.1 Adaptation

In this chapter we apply the ideas that we sketched in the previous one to improve the three nonparametric regression procedures mentioned above. Beginning with kernel estimators and local polynomial regression we provide a completely automatic and data driven technique to determine a locally defined bandwidth and their resulting approximation. More precisely, first a short overview of the basic ideas of the procedure is given, then the different steps of the method are specified, and additional information is given which is necessary for the automatic process. After providing the results for kernel estimators and local polynomials we then discuss in detail some problems that are connected with this procedure. For each of the following examples the Epanechnikov kernel is used as weighting function but any other kernel function would also be possible.

3.2 The Procedure

In our procedure of bandwidth selection we try to apply the ideas of an adequate approximation that we described in chapter 2. For kernel estimators this means, that we are looking for a bandwidth $h = (h_0 \dots h_n)$ for which the resulting data approximation $\hat{f}_h$ is an adequate approximation. As we have shown before, the residuals can be used to decide whether the approximation fulfills our requirements of adequacy or not. On the other hand it is necessary to require some kind of simplicity. Hence we are looking for an approximation of the data which satisfies the residual criteria and at the same time is as smooth as possible. Because of the bandwidth being the parameter which controls the smoothness of the regression function obviously *as smooth as possible* in the case of kernel estimators is tantamount to using a bandwidth as large as possible. Therefore we have chosen an iterative procedure which computes kernel estimates of the data and reduces the bandwidth locally depending on the result of the residual analysis.

After these general ideas we first describe the procedure of bandwidth selection in detail, show some interim approximations, and then present some approximation results and the automatically adapted bandwidth.

3.2.1 Iteration Steps

With respect to the simplicity requirement an iterative procedure of bandwidth selection is proposed. It consists of three different iteration steps. In **Step 1** the kernel estimation is computed using the current bandwidth. **Step 2** contains the multiresolution analysis of the residuals. This analysis determines on which regions the bandwidth is reduced in **Step 3**. To be sure to satisfy also the requirement of smoothness, the global constant starting value of the bandwidth is chosen so large that in each local approximation all of the data points are considered. For $0 = t_0 < \cdots < t_n = 1$ this for example would be $h_0 = h_1 = \cdots = h_n = 1$.

3.2.2 Residual Analysis on Dyadic Intervals

We have seen that our aim to split the data in a signal plus noise leads to the fact that the residuals $r_i = y_i - \hat{f}_h(t_i)$ of an approximation $\hat{f}_h$ resulting from (1.4) or (1.6) with locally defined bandwidth h have to fulfill the following condition:

$$\frac{1}{\sqrt{N_I}} |\sum_I r_i| \leq \sigma \sqrt{2\log(n)} \tag{3.1}$$

for all intervals I. One possibility is to check in **Step 2** the multiresolution condition (3.1) for every subinterval of $[t_1, t_n]$. For large data sets this can be very time consuming. Practical experience shows that the results are almost the same and that it is absolutely sufficient if we reduce our definition *to look like white noise*, to requiring the multiresolution condition on dyadic intervals as it is used in connection with the taut string (Davies and Kovac, 2001). This means that for an approximation $\hat{f}_h$ with residuals r_i, $i = 0 \ldots n$ in **Step 2** the following multiresolution coefficients are computed (for an easy description it is assumed that $n + 1 = 2^l$):

$$w_{j,k} = \frac{1}{2^{j/2}} |\sum_{i=k2^j}^{(k+1)2^j-1} r_i| \qquad j = 0, \ldots, l, \quad k = 0, \ldots, (2^{l-j} - 1). \tag{3.2}$$

For these coefficients it is checked if they satisfy

$$w_{j,k} \leq \sigma \sqrt{2\log(n)}. \tag{3.3}$$

Is $w_{j_0,k_0} > \sigma\sqrt{2\log(n)}$ the corresponding bandwidth-values are reduced in **Step 3** by multiplication with a factor α:

$$h_i^{(new)} = \alpha h_i^{(old)} \qquad i = k_0 2^{j_0}, \ldots, (k_0 + 1)2^{j_0} - 1.$$

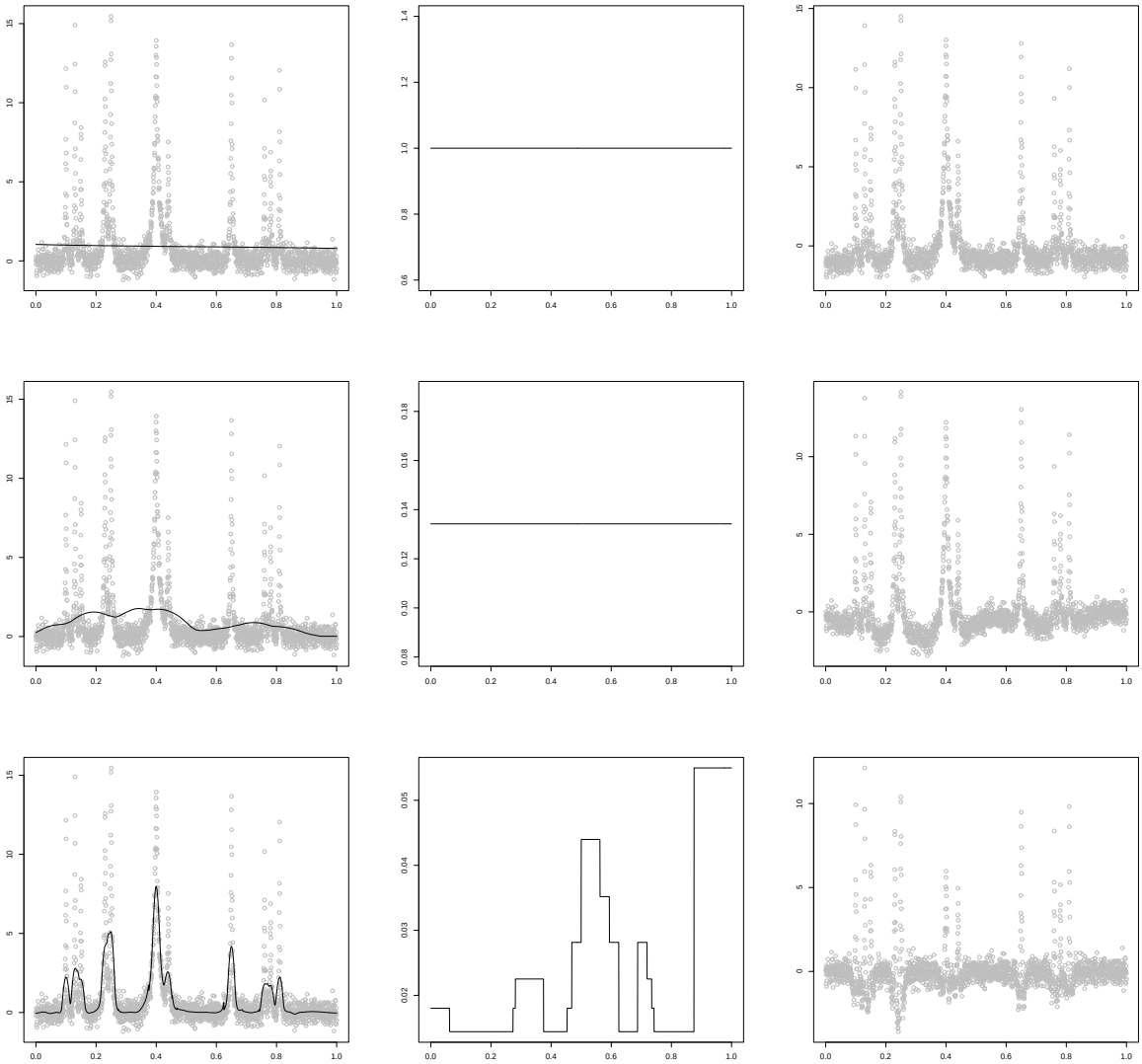


Figure 3.1: Rows: first, 10th and 20th iteration step of a kernel estimation with local MR-based adapted bandwidth. First column: data and approximation, second column: bandwidth, third column: residuals

According to the description we can now constitute the three iteration steps as:

Step 1 kernel estimation $\hat{f}_h$ for the data using bandwidth h,
Step 2 multiresolution analysis of the residuals on the dyadic intervals and
Step 3 reduction of the bandwidth where necessary.

These three steps are iterated until the number of intervals in which the multiresolution conditions are not satisfied (**Step 2**) becomes zero.

Figure 3.1 shows some steps of the process of this approximation method. The behavior of the approximation (left), bandwidth (middle) and residuals (right) of the first, 10th and 20th iteration for the Bumps data set can be seen. The very first approximation, using the constant bandwidth $h = 1.0$, is a simple line. This is obviously not satisfactory for the data set which can be seen in the residuals, which almost look like the data itself. In the procedure this decision is now given by the multiresolution criteria, the bandwidth has to be reduced. And this is also the case until the 10th iteration (second row). The bandwidth still remains constant which is a sign for the fact that in no region the multiresolution criteria have been satisfied so far. And this can also be seen in the residuals or the approximation. The peaks are not recovered at all and even the baseline is not approximated very well. But after twenty iterations (bottom) the residuals possess some regions where we would expect them to look like white noise, but still the peaks are not recovered in a satisfactory way. This impression is confirmed by the now piecewise constant bandwidth. On some intervals the decreasing process has stopped, here the bandwidth remains larger whereas on other parts it still has to be decreased.
This process is continued until all of the multiresolution criteria are satisfied and thus the residuals look like white noise. The resulting bandwidth then is piecewise constant and the approximation fulfills all of the requirements to an adequate approximation specified so far.

3.2.3 Estimation of the Noise Level

Equation (3.3) shows that there is still one parameter which has to be specified to achieve a completely automatic procedure, namely the noise level σ. It can be shown (see e. g. Davies and Kovac, 2001) that in most cases the standard deviation for the Gaussian noise can be approximated very well by

$$\sigma_n = \frac{1.48}{\sqrt{2}}\text{Median}(|y_1 - y_0|, \ldots, |y_n - y_{n-1}|). \tag{3.4}$$

For each of our examples this approximation is used, but others would also be possible.

3.2.4 Modification

In our automatic bandwidth selection procedure there are some parameters left which can be changed to influence the result, like for example the starting value of the bandwidth. We usually use the largest possible bandwidth $h = |t_n - t_0|$ which means that in our first approximation each locally computed value $\hat{f}(t_i)$, $i = 0, \ldots, n$ relies on all of the data values $y_0, \ldots, y_n$ with presumably different non-zero weights. It is possible to start with any locally defined bandwidth but in choosing its values it has to be taken into account that during the procedure the bandwidth can only be reduced but not increased. So the starting values define the largest possible values for the adapted bandwidth. But nevertheless the starting value of the bandwidth affects the computing time needed, hence the number of iterations can be smaller or larger.

The factor α used in **Step 3** controls how fast we shrink the bandwidth locally in the procedure. The default value is $\alpha = 0.8$. Our experience shows that sometimes it can be important not to use a too small value. As it is shown in Figure 3.2, the resulting bandwidth of the procedure is a piecewise constant step function. A small value in α can cause large differences in the steps of the bandwidth and sometimes artefacts in the approximation. More about these problems in section 3.3.3.

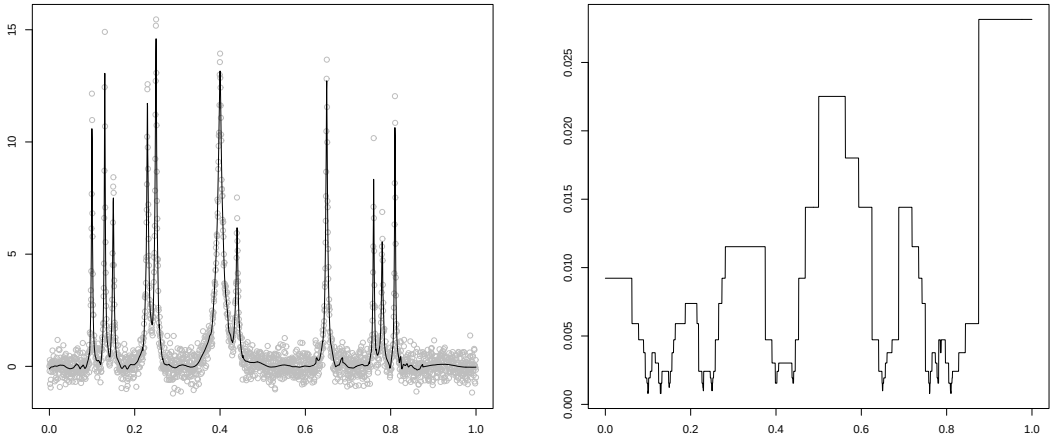


Figure 3.2: Left: result of the kernel regression with automatic MR-based bandwidth selection, Right: the adapted bandwidth.

3.3 Results

3.3.1 Kernel Estimator

For the Bumps data set we need 33 iteration steps with the default values to obtain an approximation which satisfies the multiresolution criteria on all dyadic intervals. These 33 iteration steps are computed in less than 15 seconds on an Intel Pentium 32bit processor with 256MB RAM and less than one second on a AMD Athlon 64bit processor with 512MB RAM. As it can be seen in Figure 3.2 the structure of the data has some implications on the finally used bandwidth. At the positions of the assumed peaks in the data the bandwidth becomes very small as one would expect whereas on the regions between those peaks the bandwidth remains larger. Therefore the approximating function recovers rather well the peaks and also the smooth baseline.

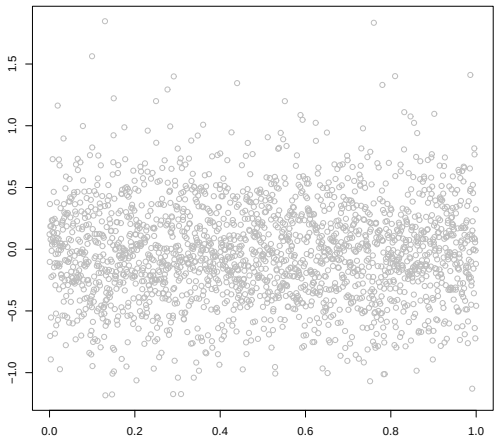


Figure 3.3: The residuals of the kernel approximation.

Finally we can also compare the resulting residuals of the Bumps kernel approximation (Figure 3.3) with the simulated normal distributed data in Figure 2.1. It might look conspicuous that there are some larger values which result from the somewhat under estimated peaks. But also these values are consistent with our multiresolution criteria and could sometimes occur in white noise.

3.3.2 Local Polynomials

It is obviously easy to transfer the idea of this bandwidth selection method to local polynomials just by replacing the kernel estimation in **Step 1** of the procedure by a local polynomial regression as it was introduced in paragraph 1.1.2. Everything else, like the multiresolution criteria and the stepwise reduction of the bandwidth remains the same.

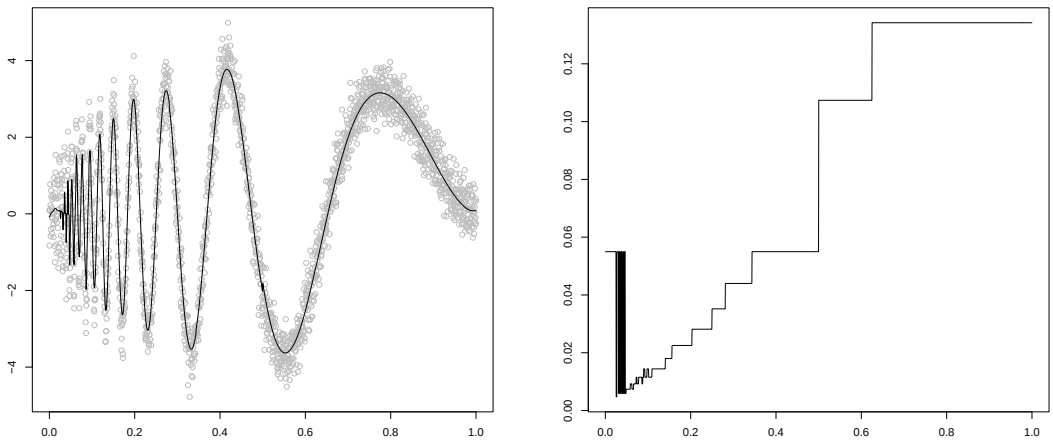


Figure 3.4: Left: result of the local polynomial regression (polynomial of order 3) with automatic MR-based bandwidth selection, right: the adapted bandwidth

The local polynomial approximation of the Doppler data (Figure 3.4) visualizes again the advantages of the bandwidth selection method based on the multiresolution analysis of the residuals. The difficulty of the Doppler data set is the adaptation of the quick oscillating sine-function in the left and a very smooth section in the right part of the data. The resulting approximation of our procedure shows both of these characteristics especially in contrast to other local bandwidth selection methods or the global bandwidth kernel approximation in Figure 2.3. In the smoother regions the bandwidth remains larger whereas the bandwidth becomes very small where more oscillations occur. If we compare the local polynomial approximation, which is very close to what we reach with the same bandwidth selection procedure and a simple kernel estimator, to the kernel approximation result in Figure 1.8, it becomes obvious that only our procedure is able to provide a smooth approximation in the right part of the data. Some of the quick oscillations might not be as distinctive as in the result of the local plug-in method but again our procedure is able to adapt more of these first oscillations.

3.3.3 Discontinuities

After emphasizing the positive properties of the procedures we also want to illustrate its difficulties. There are two different kinds of discontinuity problems related to this method. One can be seen in the local polynomial approximation of the Doppler dataset in Figure 3.4. Around 0.5 in the adapted bandwidth a very large jump occurs which causes a little artefact in the approximation. Naturally, a discontinuous bandwidth cannot be expected to lead to a continuous approximation, but as long as the jumps in the bandwidth are small enough they do not cause any problems. But larger jumps cause artefacts, in particular this can be seen in the first order differences of the Doppler approximation (Figure 3.7).

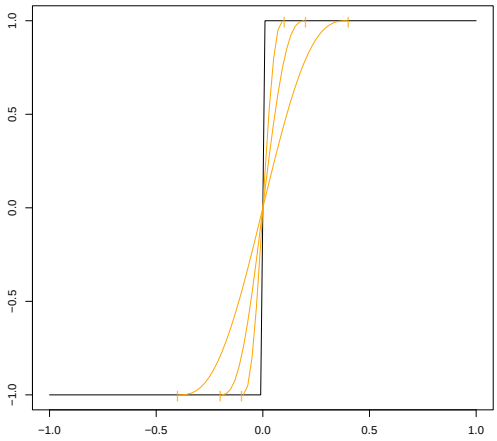


Figure 3.5: Replacing the jump in the bandwidth by a two times differentiable polynomial.

These artefacts due to the step-character of the bandwidth can be avoided by using a smooth bandwidth for the regression. One possibility which improves the result but takes some time in the computation is to smooth the chosen bandwidth afterwards, by using a simple kernel estimator with constant bandwidth. We again use an iteration procedure, in which the bandwidth of this new kernel estimator is reduced unless the resulting data approximation satisfies again—as before while using the un-smoothed bandwidth—the MR conditions. But smoothing the bandwidth as described means that from the computational point of view we have to go a step backwards because temporary we loose the property that the approximation fulfills the multiresolution criteria.
Another possibility to smooth the adapted bandwidth and to consider the multiresolution conditions at the same time is to replace the jumps by a two times differentiable polynomial. In practice this means that for each knot $p_1, \ldots, p_l$ of the bandwidth, δ_i, $i = 1, \ldots, l$ is chosen and a polynomial fitted on $[p_i - \delta_i, p_i + \delta_i]$. Then δ_i is increased stepwise, as it can be seen in Figure 3.5. In this manner we do not loose the multiresolution properties of the approximation. The δ_i can be increased as long as the multiresolution criteria are satisfied.
Using an automatic procedure to compute the smoothed version of the adapted bandwidth we obtain a local polynomial approximation to the Doppler data without artefacts. Figure 3.6 shows the smoothed bandwidth and also the approximation result. Comparing the two approximations it can be seen, that the artefact disappeared. Also the first order differences now are smoother than before (Figure 3.7).

More complicated than jumps in the bandwidth are jumps in the data as they occur in the Heavisine and the Blocks data set. It is almost impossible to get a satisfactory kernel estimation result for the Blocks data without any jump detection or something comparable. As one would expect, the adapted bandwidth at these points and their neighborhoods becomes very small. But still a small bandwidth considers data points symmetric to the

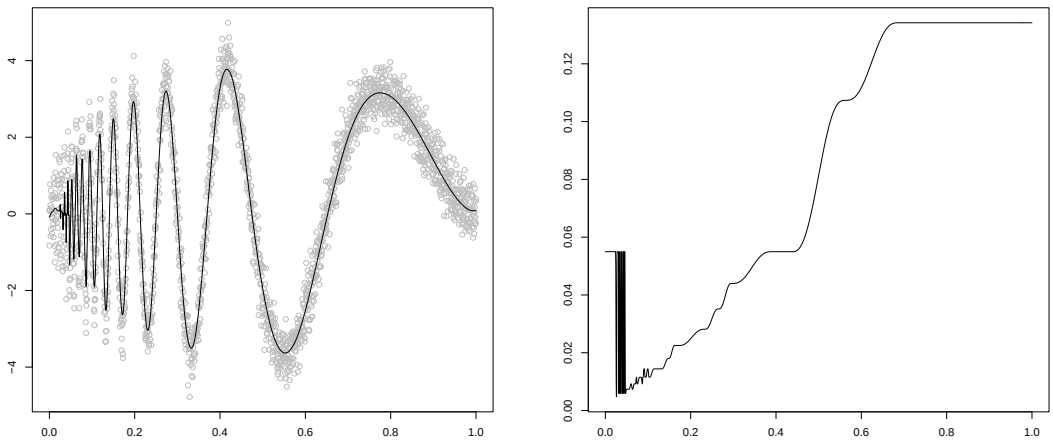


Figure 3.6: Local polynomial approximation of the Doppler data with smoothed bandwidth.

left and the right of the design point which leads to over-smoothed jumps. Figure 3.8 shows the results of our bandwidth selection technique (without bandwidth smoothing) for these two data sets. As it can be seen in the case of larger jumps the bandwidth becomes so small that almost only the design point itself influences the approximation. Only with such small values it is possible to satisfy the multiresolution conditions. But using these bandwidth values the result of the kernel approximation contains undesirable edge or boundary effects.

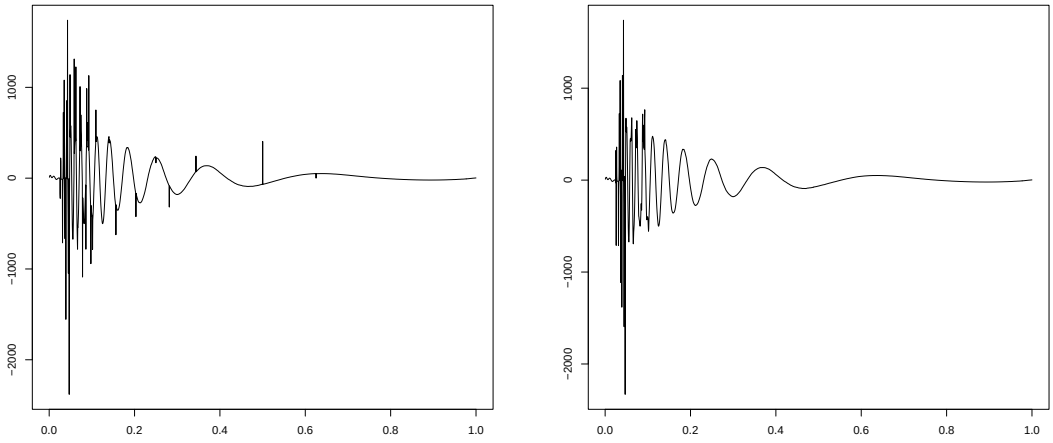


Figure 3.7: The first order differences of the local polynomial approximation of the Doppler data using the piecewise constant bandwidth (left) and the smoothed version (right).

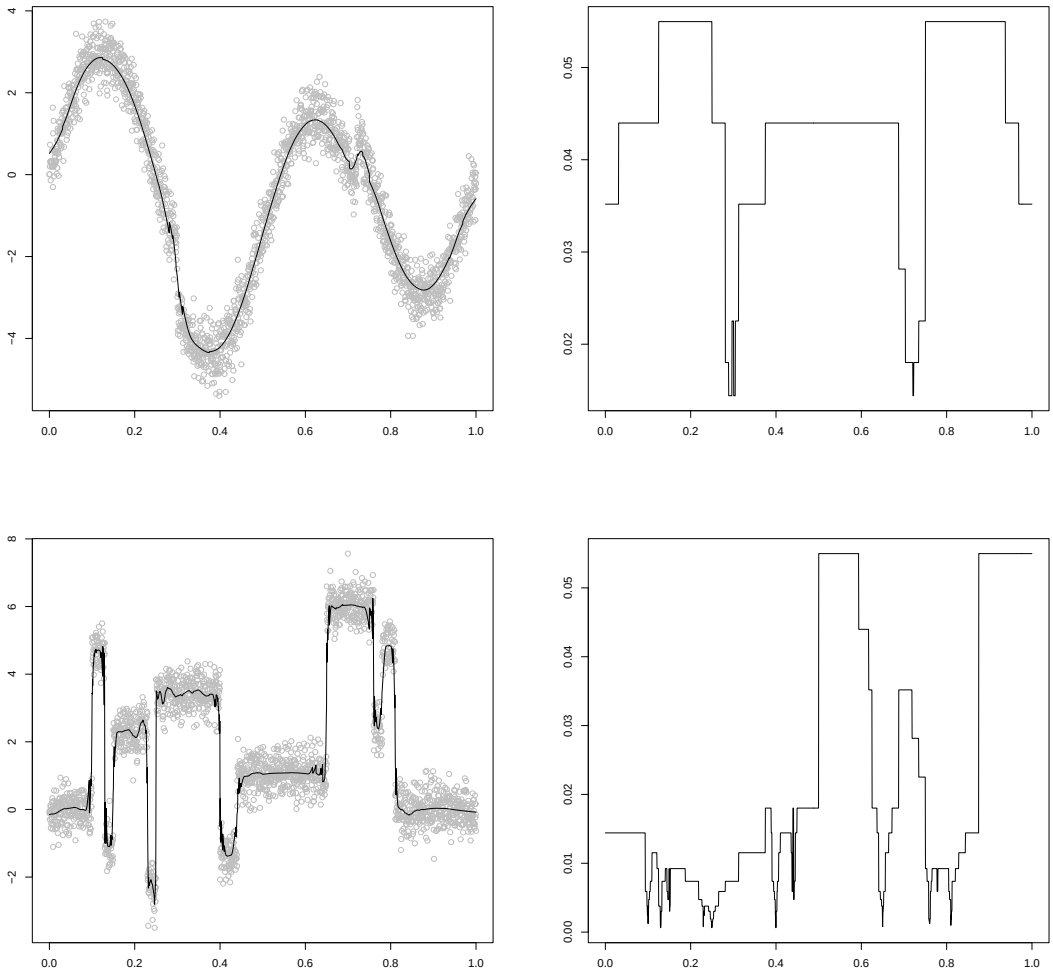


Figure 3.8: Kernel approximation and automatically adapted bandwidth for the Heavisine and the Blocks data set.

Chapter 4

Weighted Smoothing Splines

In the previous chapter a detailed description of the procedure for bandwidth selection was given. Now the idea will be adapted to the case of smoothing splines. After presenting a localized spline version we will specify the different steps of the automatic method and present the results for the four Donoho and Johnstone data sets. Finally we establish some advantages of this technique and compare the results with the adaptive weights smoothing (AWS) of Polzehl and Spokoiny (2000).

4.1 Localization

In the definition of the smoothing splines (1.9), obviously $S_\lambda(f)$ contains already both aspects of the idea of an adequate approximation. The fidelity term assures the closeness to the data. Increasing data closeness leads to rising complexity. Hence a penalty term is needed which adjusts the smoothness of the resulting spline. The penalty parameter λ controls the proportion between fidelity and penalty term. Therefore an adaptation of the multiresolution idea, which is an additional fidelity criterion, has to specify this smoothing parameter. However, the global parameter does not give enough flexibility of the approximation, as it could be seen before. A localization as in the kernel estimator case is needed. For example one localized smoothing spline version was provided by Abramovich and Steinberg (1996):

$$S_{\lambda_{\text{loc}}}(f) = \sum_{i=0}^{n}(y_i - f(t_i))^2 + \int_a^b \lambda^2(t)(f''(x))^2 dx. \tag{4.1}$$

An easier and more familiar idea to extend the possibilities of the spline approximation and to obtain more flexibility by using some kind of localization beside the possibility of knot selection is the alternative of adding local weights in the fidelity term of the penalized least squares problem:

$$S_w(f) = \sum_{i=0}^{n} w_i(y_i - f(t_i))^2 + \lambda \int_a^b (f''(x))^2 dx. \tag{4.2}$$

As described in Green and Silverman (1993), there exists a unique minimizer $\hat{f}_w$ of $S_w(f)$, which is a natural cubic spline with knots in t_i. It is easy to modify the Reinsch algorithm, mentioned above, by incorporating the diagonal weight matrix. Hence almost the same computations for the solution can be used as in the non-weighted case.

By a skilful choice of the w_i it is possible to improve the local adaptivity of the smoothing spline approximation. De Boor (2001) points out that there exists no complete automatic software for the determination of locally weighted smoothing splines and that they do not reach a great deal of attention yet. We now show how the multiresolution conditions can be used successfully for an automatic determination of the local weights for weighted smoothing splines.

4.2 Procedure and Regression Results

The local weights in case of the smoothing splines (4.2) are located in the fidelity term of the minimization problem. Hence they specify directly how close the local approximation value will be to the corresponding data point. Therefore large weighting values force the adapted value to be close to the data. Whereas small values lead to a smoother approximation.
To obtain an approximation which is as smooth as possible and which fulfills the multiresolution criteria, we have to start with very small local weights which will be increased during the iteration.
Obviously, an automatic determination of the weights implicitly defines a smoothing parameter λ, because

$$\begin{aligned}
\hat{f}_w &= \arg\min_f S_w(f) \\
&= \arg\min_f \frac{1}{\lambda} S_w(f) \\
&= \arg\min_f \left(\sum_{i=0}^{n} \frac{w_i}{\lambda} (y_i - f(t_i))^2 + \int_a^b (f''(x))^2 dx \right).
\end{aligned}$$

Even if standard software for the computation of smoothing splines provides a possibility to include local weights, then the smoothing parameter λ still has to be determined. An automatic procedure to determine the local weights avoids this additional specification.

The algorithm for the locally weighted smoothing spline approximation contains almost the same steps as in the kernel regression methods:

Step 1 computing the solution of the minimization problem (4.2)
by using weights $w = w_0 \dots w_n$,
Step 2 multiresolution analysis of the residuals on the dyadic intervals and
Step 3 increasing the weights where necessary.

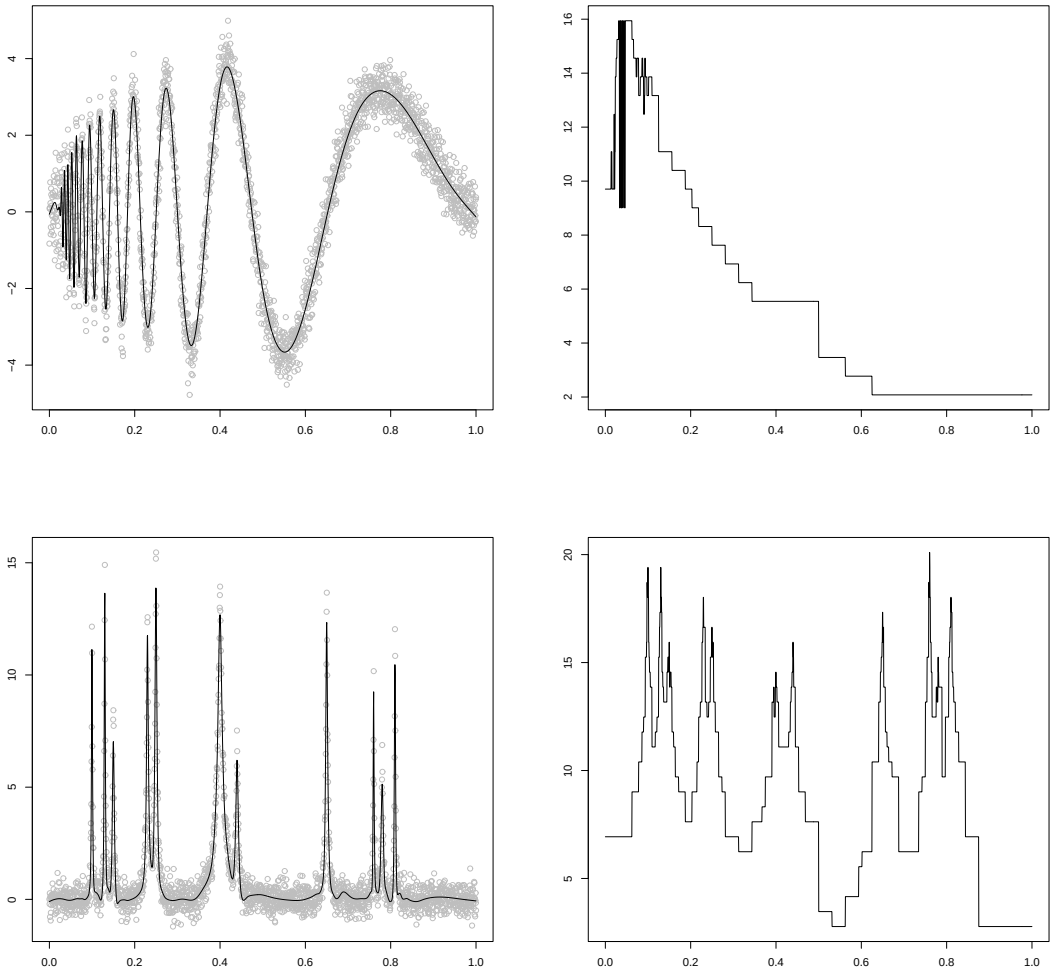


Figure 4.1: Weighted splines approximation and its MR-based adapted local weights plotted on logarithmic scale.

The iteration stops once the multiresolution conditions are satisfied on all dyadic intervals. A default value of $\alpha = 2$ is used to increase the local weights in **Step 3**, the starting values for the weights w_i are chosen dependent on the data, for $t_i = \frac{i}{n}$ this would be $w_0 = \ldots_= w_n = (n+1)^{-1}$. Using larger starting values would avoid some iteration steps but on the other hand in some cases the approximation would not be as smooth as it could be.

Figure 4.1 and 4.2 show the approximations for all four Donoho-Johnstone data sets. The locally adapted weights are plotted on the right column using a logarithmic scale. As it can be seen, the weighted smoothing spline is able to catch most of the fast oscillations in the left part of the dataset and still remains smooth in the right. This is reflected

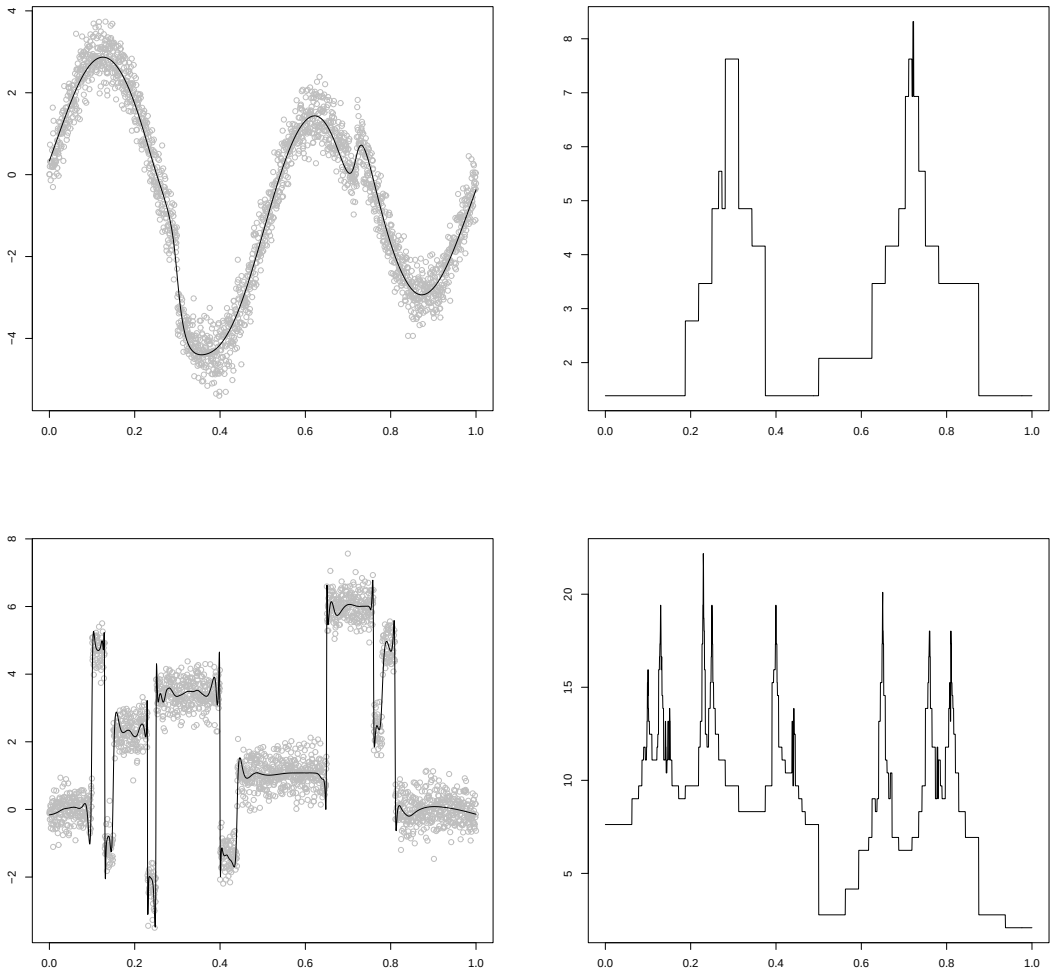


Figure 4.2: Weighted splines approximation of the two discontinuous test datasets and its MR-based locally adapted weights, plotted on logarithmic scale. In the blocks approximation (bottom left) the Gibbs effect becomes obvious.

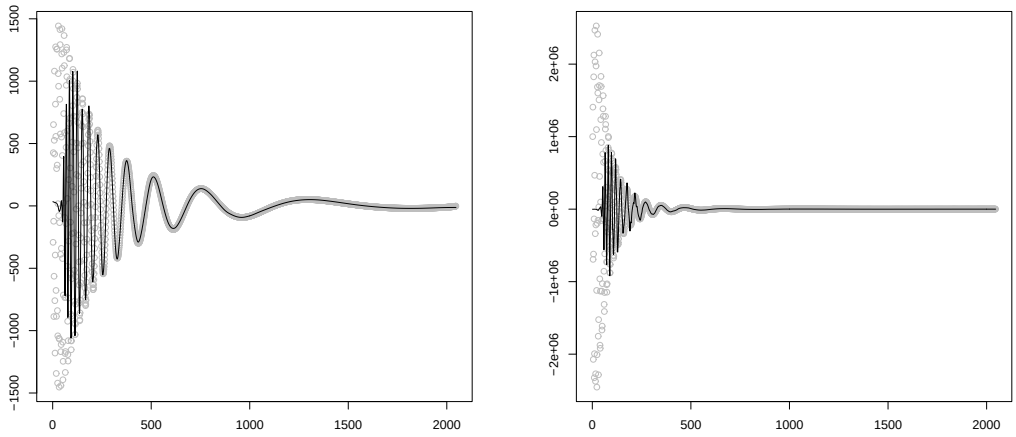


Figure 4.3: The real first (left) and second (right) derivative of the Doppler function (grey points) and its approximation by the first and second order differences of the weighted spline adaptation of the Doppler dataset.

in the corresponding weights, which become very large near these rapid oscillations. The procedure also permits a reasonable approximation of the thin bump features in the second dataset with very smooth parts in between.
In contrast to the kernel regression case, the piecewise constant structure of the weights does not cause any problems since the solution of the minimization problem is still given by a normal cubic spline. Hence no additional smoothing techniques are necessary.
Another point are the discontinuous datasets, or better the data sets of discontinuous functions, the Heavisine and the Blocks. Since the solution of the minimization problem is differentiable we cannot expect the smoothing spline to cope with those datasets. Therefore different fidelity or penalty terms would be needed. A cubic smoothing spline will always over smooth discontinuouities as it can be seen in Figure 4.2. In the case of large jumps in the data the spline tends to overshoot at the edges. This phenomenon is known as the Gibbs effect.

4.3 Derivative Estimation

Since splines are twice continuously differentiable, they are particularly suitable for a simple way to estimate the derivative. The example of the Doppler approximation using weighted splines shows that they provide such good results that also the first order differences of the adapted values generate a more than acceptable approximation of the first derivative of the Doppler function (see Figure 4.3). And except for the first part which contains the high oscillating data and function part also the second order differences adapt the real second derivative very well.

One procedure which also gives very good approximation results, but does not have the advantage of the differentiability, is the adaptive weight smoothing of Polzehl and Spokoiny (2000), which provides a local approximation method for one and two dimensions. Here the data are approximated locally, using a polynomial of degree $p \leq 3$. Similar to our procedure the considered interval is increased stepwise. But in the AWS case each new local estimation $\hat{f}_k(t_j)$ is compared with the previous ones $\hat{f}_{k'}(t_j)$ for all $k' < k$. If $|\hat{f}_k(t_j) - \hat{f}_{k'}(t_j)| > s(t_j)$ for only one $k' < k$ and some specified local threshold $s(t_j)$ the increasing process stops and the last accepted estimate is used.
Figure 4.4 shows the result of the approximation using $p = 3$, which is similar to the weighted smoothing spline result, except for some outliers. But as the first order differences of the AWS result show in differentiable cases one would prefer the approximation splines.

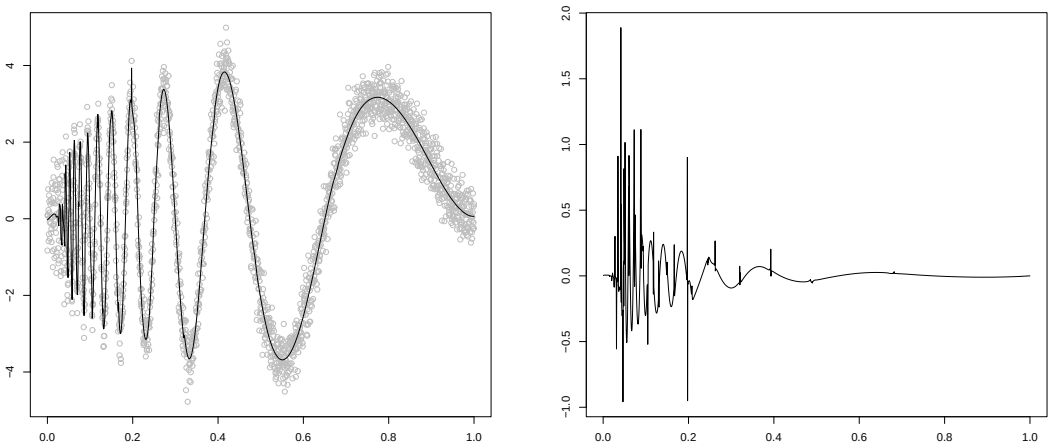


Figure 4.4: Right: approximation of the Doppler data using AWS. Right: The resulting first order differences.

4.4 Asymptotics

After showing the practical results of our weighted smoothing spline method we want to mention some aspects of its asymptotic convergence. Because of the way of choosing the local weights in our procedure it is difficult to prove some general asymptotics, but still we can state a proposition concerning the general behavior of weighted smoothing splines. Of course the well known asymptotic behavior of the general cubic smoothing spline (cf. Craven and Wahba (1979)) also holds for the weighted version (4). The proof shown here is similar to the proposed one, which was supplemented by Utreras (1983), varied and expanded in several other publications.

Notation:
For an equidistant grid ξ_n with grid points $t_i = \frac{i}{n-1}$, $i = 1, \ldots, n$ and a function f on $[0, 1]$

we write

$$\mathbf{f} = \begin{pmatrix} \mathbf{f}_1 \\ \mathbf{f}_2 \\ \vdots \\ \mathbf{f}_n \end{pmatrix} = \begin{pmatrix} f(t_1) \\ f(t_2) \\ \vdots \\ f(t_n) \end{pmatrix}.$$

S_n^m means the linear space of natural splines of order $2m-1$.
From the basic spline theory and a special paper of Utreras (1983) we know the following:

Theorem 4.1. *For $\mathbf{y} \in \mathbb{R}^n$ there exists a unique function $g \in S_n^m$, so that $g(t_i) = y_i$, $i = 1, \ldots, n$ and the transformation $\mathbf{y} \to g$ is linear. Then there exists a real positive semi-definite matrix Ω with*

$$\int_0^1 g^{(m)}(t)^2 \, dt = < \mathbf{y}, \Omega \mathbf{y} >$$

and $<,>$, the scalar product on $\mathbb{R}^n$. Let $\alpha_1, \ldots, \alpha_n$ be the eigenvalues of Ω. Then $\alpha_1 = \cdots = \alpha_m = 0$ and there exist constants c_1 and c_2 so that

$$\frac{1}{n} c_1 i^{2m} \leq \alpha_i + m \leq \frac{1}{n} c_2 i^{2m} \qquad i = 1, \ldots, n-m. \tag{4.3}$$

Using these preconditions we are able to show the following for simple smoothing splines known theorem.

Theorem 4.2. *Let $(\xi_n)_{n \in \mathbb{N}}$ be a sequence of equidistant grids of length n and f_n the solution of the minimization problem*

$$\min_g S_\lambda(g) = \min_g \sum_{i=1}^n w_i (y(t_i) - g(t_i))^2 + \lambda \int (g''(u))^2 \, du \tag{4.4}$$

with $y(t_i) = f(t_i) + \varepsilon_i$, $\varepsilon_i \sim N(0,1)$ and arbitrary but fixed w_i, so that $0 < w_{\min} \leq w_i \leq w_{\max} < \infty$ $i = 1, \ldots, n$. Then there exist constants $C_1, C_2 \in \mathbb{R}$ so that

$$\frac{1}{n} \mathbb{E}(\|\mathbf{f_n} - \mathbf{f}\|_2^2) \leq C_1 \frac{\lambda}{n} + \sigma^2 C_2 n^{-\frac{3}{4}} \lambda^{-\frac{1}{4}}. \tag{4.5}$$

Proof:
We define the diagonal matrix $W := \operatorname{diag}(w_1, \ldots, w_n)$ and write (4.4) as

$$\begin{aligned} S_w(g) &= (\mathbf{y} - \mathbf{g})^T W (\mathbf{y} - \mathbf{g}) + \lambda \mathbf{g}^T \Omega \mathbf{g} \\ &= \mathbf{y}^T W \mathbf{y} - 2\mathbf{g}^T W \mathbf{y} + \mathbf{g}^T W \mathbf{g} + \lambda \mathbf{g}^T \Omega \mathbf{g} \end{aligned}$$

For the solution f_n of the minimization problem we have:

$$2W\mathbf{f_n} - 2W\mathbf{y} - 2\lambda\Omega\mathbf{f_n} = 0$$

and hence

$$\begin{aligned}\mathbf{f_n} &= (W+\lambda\Omega)^{-1}W\mathbf{y}\\ &= (W+\lambda\Omega)^{-1}W(\mathbf{f}+\varepsilon)\\ &= (W+\lambda\Omega)^{-1}(W+\lambda\Omega)\mathbf{f} - \lambda(W+\lambda\Omega)^{-1}\Omega\mathbf{f} + (W+\lambda\Omega)^{-1}W\varepsilon.\end{aligned}$$

We obtain $\mathbf{f_n} - \mathbf{f} = -\lambda(W+\lambda\Omega)^{-1}\Omega\mathbf{f} + (W+\lambda\Omega)^{-1}W\varepsilon$

With $A \ll B \quad \Leftrightarrow \quad \mathbf{x}^T A\mathbf{x} \leq \mathbf{x}^T B\mathbf{x} \quad \forall\mathbf{x}$ we have

$$(w_{\min}I + \lambda\Omega) \ll (W+\lambda\Omega) \ll (w_{\max}I + \lambda\Omega)$$

and thus

$$(w_{\max}I + \lambda\Omega)^{-1} \ll (W+\lambda\Omega)^{-1} \ll (w_{\min}I + \lambda\Omega)^{-1}.$$

A basis of eigenvectors $\mathbf{v_i}$ can be found so that $\mathbf{f}$ can be written as $\mathbf{f} = \sum_{i=1}^n \gamma_i\mathbf{v_i}$ and $\varepsilon = \sum_{i=1}^n Z_i\mathbf{v_i}$ with Z_i i.i.d. $N(0,\sigma^2)$. Furthermore we have

$$\begin{aligned}(w_{\min}I + \lambda\Omega)^{-1}(w_{\min}I + \lambda\Omega)\mathbf{v_i} &= \mathbf{v_i}\\ (w_{\min}I + \lambda\Omega)^{-1}(\mathbf{v_i} + \lambda\alpha_i\mathbf{v_i}) &= \mathbf{v_i}\\ (w_{\min}I + \lambda\Omega)^{-1}\mathbf{v_i} &= \frac{1}{w_{\min}+\lambda\alpha_i}\mathbf{v_i}.\end{aligned}$$

Therefore we obtain

$$\mathbb{E}(\|\mathbf{f_n} - \mathbf{f}\|_2^2) \leq \sum_{i=1}^n \gamma_i^2 \frac{\lambda^2\alpha_i^2}{(w_{\min}+\lambda\alpha_i)^2} + \sum_{i=1}^n \sigma^2 \frac{w_{\max}^2}{(w_{\min}+\lambda\alpha_i)^2}.$$

Using (4.3) we find the following estimates for the two parts of the expectation:

$$\begin{aligned}
\sum_{i=1}^{n} \frac{w_{\max}^2}{(w_{\min}+\lambda\alpha_i)^2} &= \sum_{i=3}^{n} \frac{w_{\max}^2}{(w_{\min}+\lambda\alpha_i)^2} + 2\frac{w_{\max}^2}{w_{\min}^2} \\
&= \sum_{i=1}^{n-2} \frac{w_{\max}^2}{(w_{\min}+\lambda\alpha_{i+2})^2} + 2\frac{w_{\max}^2}{w_{\min}^2} \\
&\le \sum_{i=1}^{n-2} \frac{w_{\max}^2}{(w_{\min}+\lambda\frac{c_1}{n}i^4)^2} + 2\frac{w_{\max}^2}{w_{\min}^2} \\
&\le \sum_{i=1}^{\lfloor n^{\frac{1}{4}}\lambda^{-\frac{1}{4}}\tilde{c}_1^{-\frac{1}{4}}\rfloor} \frac{w_{\max}^2}{w_{\min}^2} + \sum_{i=\lceil n^{\frac{1}{4}}\lambda^{-\frac{1}{4}}\tilde{c}_1^{-\frac{1}{4}}\rceil}^{n-2} \frac{w_{\max}^2 n^2}{\lambda^2 c_1^2 i^8} + 2\frac{w_{\max}^2}{w_{\min}^2} \\
&\le \frac{w_{\max}^2}{w_{\min}^2} n^{\frac{1}{4}}\lambda^{-\frac{1}{4}}\tilde{c}_1^{-\frac{1}{4}} + \frac{w_{\max}^2 n^2}{\lambda^2}(n^{\frac{1}{4}}\lambda^{-\frac{1}{4}}\tilde{c}_1^{-\frac{1}{4}})^{-8+1} + 2\frac{w_{\max}^2}{w_{\min}^2} \\
&\le n^{\frac{1}{4}}\lambda^{-\frac{1}{4}}\tilde{\tilde{c}}_1 + n^{\frac{1}{4}}\lambda^{-\frac{1}{4}}\tilde{\tilde{c}}_1 + 2\frac{w_{\max}^2}{w_{\min}^2} \\
&\le C_2\, n^{\frac{1}{4}}\lambda^{-\frac{1}{4}} + 2\frac{w_{\max}^2}{w_{\min}^2}
\end{aligned}$$

and

$$\begin{aligned}
\sum_{i=1}^{n} \gamma_i^2 \frac{\lambda^2\alpha_i^2}{(w_{\min}+\lambda\alpha_i)^2} &= \sum_{i=3}^{n} \gamma_i^2 \frac{\lambda^2\alpha_i^2}{(w_{\min}+\lambda\alpha_i)^2} \\
&= \sum_{i=1}^{n-2} \gamma_{i+2}^2 \frac{\lambda^2\alpha_{i+2}^2}{(w_{\min}+\lambda\alpha_{i+2})^2} \\
&\le \sum_{i=1}^{n-2} \gamma_{i+2}^2 \frac{\lambda^2\frac{c_2^2}{n^2}i^8}{(w_{\min}+\lambda\frac{c_1}{n}i^4)^2} \\
&\le \frac{\lambda^2 c_2^2}{w_{\min}^2 n^2} \sum_{i=1}^{\lfloor n^{\frac{1}{4}}\lambda^{-\frac{1}{4}}\tilde{c}_1^{-\frac{1}{4}}\rfloor} \gamma_{i+2}^2 i^8 + \frac{\lambda^2}{n^2} \sum_{i=\lceil n^{\frac{1}{4}}\lambda^{-\frac{1}{4}}\tilde{c}_1^{-\frac{1}{4}}\rceil}^{n-2} \gamma_{i+2}^2 \frac{i^8 n^2}{\lambda^2\tilde{c}_1^2 i^8} \\
&\le \frac{\lambda^2}{n^2}\tilde{c}_2^2 \sum_{i=1}^{\lfloor n^{\frac{1}{4}}\lambda^{-\frac{1}{4}}\tilde{c}_1^{-\frac{1}{4}}\rfloor} \gamma_{i+2}^2 i^8 + \tilde{c}_1^{-2} \sum_{i=\lceil n^{\frac{1}{4}}\lambda^{-\frac{1}{4}}\tilde{c}_1^{-\frac{1}{4}}\rceil}^{n-2} \gamma_{i+2}^2 \\
&\le \frac{\lambda^2}{n^2}\tilde{c}_2^2 \sum_{i=1}^{\lfloor n^{\frac{1}{4}}\lambda^{-\frac{1}{4}}\tilde{c}_1^{-\frac{1}{4}}\rfloor} \gamma_{i+2}^2 i^4 i^4 + \tilde{c}_1^{-2} \frac{(n^{\frac{1}{4}}\lambda^{-\frac{1}{4}}\tilde{c}_1^{-\frac{1}{4}})^4}{(n^{\frac{1}{4}}\lambda^{-\frac{1}{4}}\tilde{c}_1^{-\frac{1}{4}})^4} \sum_{i=\lceil n^{\frac{1}{4}}\lambda^{-\frac{1}{4}}\tilde{c}_1^{-\frac{1}{4}}\rceil}^{n-2} \gamma_{i+2}^2
\end{aligned}$$

$$\begin{aligned} &\leq \frac{\lambda^2}{n^2} n\lambda^{-1}\hat{c} \sum_{i=1}^{\lfloor n^{\frac{1}{4}}\lambda^{-\frac{1}{4}}\tilde{c}_1^{-\frac{1}{4}}\rfloor} \gamma_{i+2}^2 i^4 + \bar{c}\frac{\lambda}{n} \sum_{i=\lceil n^{\frac{1}{4}}\lambda^{-\frac{1}{4}}\tilde{c}_1^{-\frac{1}{4}}\rceil}^{n-2} \gamma_{i+2}^2 i^4 \\ &\leq \hat{c}\frac{\lambda}{n} \sum_{i=1}^{\lfloor n^{\frac{1}{4}}\lambda^{-\frac{1}{4}}\tilde{c}_1^{-\frac{1}{4}}\rfloor} \gamma_{i+2}^2 i^4 + \bar{c}\frac{\lambda}{n} \sum_{i=\lceil n^{\frac{1}{4}}\lambda^{-\frac{1}{4}}\tilde{c}_1^{-\frac{1}{4}}\rceil}^{n-2} \gamma_{i+2}^2 i^4. \end{aligned}$$

Since $\sum_{i=1}^n \gamma_i^2\alpha_i = \mathbf{f}^T\Omega\mathbf{f} \leq C$ we get

$$C \geq \sum_{i=1}^n \gamma_i^2\alpha_i \;=\; \sum_{i=3}^n \gamma_i^2\alpha_i \geq \sum_{i=1}^{n-2} \gamma_{i+2}^2 \frac{c_i}{n} i^4$$

and hence $\sum_{i=1}^{n-2} \gamma_{i+2}^2 i^4 \leq n\tilde{c}$. Thus we obtain

$$\sum_{i=1}^n \gamma_i^2 \frac{\lambda^2\alpha_i^2}{(w_{\min}+\lambda\alpha_i)^2} \;\leq\; \frac{\lambda}{n}\bar{C}n + \frac{\lambda}{n}\bar{\bar{C}}n \leq \tilde{c}_1\lambda$$

Concluding we have

$$\begin{aligned} \frac{1}{n}\mathbb{E}(\|\mathbf{f_n}-\mathbf{f}\|_2^2) &\leq \frac{1}{n}(C_1\lambda + \sigma^2(C_2 n^{\frac{1}{4}}\lambda^{-\frac{1}{4}} + C_3)) \\ &\leq C_1\frac{\lambda}{n} + \sigma^2 C_2 n^{-\frac{3}{4}}\lambda^{-\frac{1}{4}} + \sigma^2\frac{C_3}{n}. \end{aligned}$$

□

Corollary 4.1. *With the assumptions of Theorem 4.2, for $\lambda \approx n^{\frac{1}{5}}$ we have*

$$\frac{1}{n}\mathbb{E}(\|\mathbf{f_n}-\mathbf{f}\|_2^2) = O(n^{-\frac{4}{5}}).$$

Proof:

$$\begin{aligned} \frac{1}{n}\mathbb{E}(\|\mathbf{f_n}-\mathbf{f}\|_2^2) &\leq C_1\frac{\lambda}{n} + \sigma^2(C_2 n^{-\frac{3}{4}}\lambda^{-\frac{1}{4}} + C_3 n^{-1}) \\ &\leq C_1 n^{-\frac{4}{5}} + \sigma^2 C_2 n^{-\frac{3}{4}} n^{-\frac{1}{20}} + \tilde{C}_3 n^{-1} \\ &\leq C n^{-\frac{4}{5}} + \tilde{C}_3 n^{-1}. \end{aligned}$$

□

Our weighted smoothing spline approximation is a special one in the sense that it satisfies the multiresolution conditions. We now want to use these sum conditions to prove another asymptotic behavior. Notation and procedure are similar to the one of Majidi (2003).

Definition 4.1. *Let $\xi = \xi_n$ be an equidistant grid with grid points $t_i = \frac{i}{n}$. We say, g satisfies the sum condition for ξ_n and τ, i. e. $g \in S(\xi_n, \tau)$, if and only if*

$$|\sum_{i \in \xi \cap I} g_i| \leq \sqrt{l_n}\sqrt{\tau \log(n+1)}.$$

Here $l_n = l_n(I) = \#\{t_i | 0 \leq i \leq n\} \cap I$ for any interval $I \subset [0,1]$.

Definition 4.2. *For $K > 0$ we define*

$$W_2(K) := \{f \mid f : [0,1] \to \mathbb{R},\ f \in C^2([0,1]), \sqrt{\int_0^1 f''(t)^2\, dt} \leq K\}$$

and for $f \in W_2(K)$

$$\|f''\|_2 := \sqrt{\int_0^1 f''(t)^2\, dt} \qquad \text{and} \qquad \|f\|_\infty := \sup_{x \in [0,1]} |f(x)|.$$

We now can state a proposition about the asymptotic convergence of a weighted spline approximation f_n for data $y(t_i) = f(t_i) + \varepsilon_i$, ε_i i. i. d. $N(0,1)$. For $K_1 > 0$ and $f \in W_2(K)$ we assume that we find $K_2 > 0$ so that $f_n \in W_2(K_2)$ for each n. The cubic spline, which interpolates f at the grid points t_i, is one of the funcitons which satisfies the sum conditions with high probability. Hence it will be considered in the minimization problem and therefore the assumption seems to be justified.

Theorem 4.3. *For $K_1, K_2 > 0$ let f be a function with $f \in W_2(K_1)$ with $y(t) = f(t) + \varepsilon(t)$, and $\varepsilon(t) \sim N(0,1)$. Moreover let $(\xi_n)_{n \in \mathbb{N}}$ be a sequence of equidistant grids and let f_n be the result of the weighted smoothing spline procedure so that for one $\tau > 2$ and each n $f_n - y \in S(\xi_n, \tau)$ and $f_n \in W_2(K_2)$. Then for each $\alpha > \frac{1}{2}$ and each $\delta > 0$ we have*

$$\lim_{n \to \infty} \mathbb{P}\left(n^{\frac{3}{8}} \left(\log(n+1)\right)^{-\alpha} \|f - f_n\|_\infty \leq \delta\right) = 1.$$

Proof: Setting $K := K_1 + K_2$ we can state $\|f'' - f_n''\|_2 \leq \|f''\|_2 + \|f_n''\|_2 \leq K$.
According to the hypotheses $f_n - y \in S(\xi_n, \tau)$. With increasing probability also $f - y \in S(\xi_n, \tau)$ (cf. Chapter 2). Assuming $f - y \in S(\xi_n, \tau)$ we then have

$$h_n := f - f_n = f - y - (f_n - y) \in S(\xi_n, 4\tau).$$

We set $\mu_n := n^{-\tilde{q}}$, $\frac{1}{4} < \tilde{q} < 1$, $I_n(t) := [t - \mu_n, t + \mu_n] \cap [0,1]$, $l_n := \#\{t_i \in I_n\}$ and choose $j_n \in \mathbb{N}$ so that $(j_n + 1)/n = \min_{t_i \in I_n}(t_i)$. Then

$$\{t_i, i = 0, \ldots,\} \cap I_n = \left\{\frac{j_n + 1}{n}, \ldots, \frac{j_n + l_n}{n}\right\}.$$

For $k \in \mathbb{Z}$ so that $1 \leq i+k \leq n$ we have:

$$h_n(t_{i+k}) = h_n(t_i) + (t_{i+k} - t_i)h_n'(t_i) + \int_{t_i}^{t_{i+k}} (t_{i+k} - x)h_n''(x)dx.$$

Hence we can write

$$\begin{aligned}
&|\sum_{j_n+1}^{j_n+l_n} h_n'(t_i)(t_{i+k} - t_i)| \\
&= |\sum_{i=j_n+1}^{j_n+l_n} h_n(t_{i+k}) - \sum_{i=j_n+1}^{j_n+l_n} h_n(t_i) - \sum_{i=j_n+1}^{j_n+l_n} \int_{t_i}^{t_{i+k}} (t_{i+k} - x)h_n''(x)\,dx| \\
&\leq 2\sqrt{l_n}\sqrt{4\tau\log(n+1)} + \sum_{i=j_n+1}^{j_n+l_n} |\int_{t_i}^{t_{i+k}} (t_{i+k} - x)h_n''(x)dx| \\
&\leq 2\sqrt{l_n}\sqrt{4\tau\log(n+1)} + \sum_{i=j_n+1}^{j_n+l_n} |\sqrt{\int_{t_i}^{t_{i+k}} (t_{i+k} - x)^2\,dx}\sqrt{\int_{t_i}^{t_{i+k}} (h_n''(x))^2\,dx}| \\
&\leq 2\sqrt{l_n}\sqrt{4\tau\log(n+1)} + \sum_{i=j_n+1}^{j_n+l_n} \frac{1}{\sqrt{3}}(t_{i+k} - t_i)^{\frac{3}{2}}K \\
&\leq 2\sqrt{l_n}\sqrt{4\tau\log(n+1)} + \frac{l_n k^{\frac{3}{2}}}{3n^{\frac{3}{2}}}K.
\end{aligned}$$

Now we choose n so large that $[t+\mu_n, t+3\mu_n] \subset [0,1]$ or $[t-3\mu_n, t-\mu_n] \subset [0,1]$. Then (because of $l_n/n \leq 2\mu_n$) we can choose $k_n = l_n$ or $k_n = -l_n$ so that

$$1 \leq j_n + 1 + k_n,\ j_n + l_n + k_k \leq n.$$

Continuing for $t \in (0,1)$ we can write

$$h_n(t) = h_n(t_i) + h_n'(t_i)(t - t_i) + \int_{t_i}^{t} h_n''(x)(t-x)\,dx.$$

Hence

$$\begin{aligned}
&l_n|h_n(t)| \\
&\leq |\sum_{i=j_n+1}^{j_n+l_n} h_n(t_i)| + |\sum_{i=j_n+1}^{j_n+l_n} h_n'(t_i)(t - t_i)| + |\sum_{i=j_n+1}^{j_n+l_n} \int_{t_i}^{t} h_n''(x)(t-x)\,dx| \\
&\leq \sqrt{l_n}\sqrt{4\tau\log(n+1)} + |I_n||\sum_{i=j_n+1}^{j_n+l_n} h_n'(t_i)| + \sum_{i=j_n+1}^{j_n+l_n} |\sqrt{\int_{t_i}^{t} (t-x)^2\,dx}||\sqrt{\int_{t_i}^{t} (h_n''(x))^2, dx}| \\
&\leq \sqrt{l_n}\sqrt{4\tau\log(n+1)} + |I_n||\frac{n}{k_n}|\left(2\sqrt{l_n}\sqrt{4\tau\log(n+1)}| + \frac{K}{\sqrt{3}}l_n|\frac{k_n}{n}|^{\frac{3}{2}}\right) + \sum_{i=j_n+1}^{j_n+l_n} \frac{|t-t_i|^{\frac{3}{2}}}{\sqrt{3}}K \\
&\leq \sqrt{l_n}\sqrt{4\tau\log(n+1)} + \frac{2|I_n|n}{\sqrt{l_n}}\sqrt{4\tau\log(n+1)} + l_n^{3/2}\frac{|I_n|}{\sqrt{3n}}K + l_n\frac{|I_n|^{\frac{3}{2}}}{\sqrt{3}}K.
\end{aligned}$$

Dividing by l_n and using $|I_n| \leq 2\mu_n$ and $\mu_n n \leq l_n \leq 2\mu_n n$ we get

$$\begin{aligned} |h_n(t)| &\leq l_n^{-\frac{1}{2}}\sqrt{4\tau\log(n+1)} + \frac{4\mu_n n}{l_n^{\frac{3}{2}}}\sqrt{4\tau\log(n+1)} + l_n^{\frac{1}{2}}\frac{2\mu_n}{\sqrt{3n}}K + \frac{K}{\sqrt{3}}(2\mu_n)^{\frac{3}{2}} \\ &\leq \frac{1}{\sqrt{\mu_n n}}\sqrt{4\tau\log(n+1)} + \frac{4}{\sqrt{\mu_n n}}\sqrt{4\tau\log(n+1)} + K\sqrt{\frac{8}{3}}\mu_n^{\frac{3}{2}} + K\sqrt{\frac{8}{3}}\mu_n^{\frac{3}{2}} \end{aligned}$$

With $\mu_n = n^{-\frac{1}{4}}$ we then have

$$|h_n(t)| \leq 5n^{-\frac{3}{8}}\sqrt{4\tau\log(n+1)} + 4n^{-\frac{3}{8}}K\sqrt{\frac{2}{3}}.$$

Hence for $\alpha > \frac{1}{2}$ we get

$$\begin{aligned} n^{\frac{3}{8}}\left(\log(n+1)\right)^{-\alpha}|h_n(t)| &\leq 5\sqrt{4\tau}\left(\log(n+1)\right)^{-(\alpha-\frac{1}{2})} + 4\left(\log(n+1)\right)^{-\alpha}K\sqrt{\frac{2}{3}} \\ &\longrightarrow 0 \quad (n \longrightarrow \infty). \end{aligned}$$

That means that for each $\delta < 0$, it exists $N_\delta \in \mathbb{N}$ so that for each $n \geq N_\delta$:

$$n^{\frac{3}{8}}\left(\log(n+1)\right)^{-\alpha}\|h_n\|_\infty \leq \delta.$$

Thus we have

$$\mathbb{P}\left(\{f - y \in S(\xi_n, \tau)\}\right) \leq \mathbb{P}\left(n^{\frac{3}{8}}\left(\log(n+1)\right)^{-\alpha}\|f - f_n\|_\infty \leq \delta\right)$$

and hence

$$\mathbb{P}\left(n^{\alpha}\left(\log(n+1)\right)^{-\alpha}\|f - f_n\|_\infty \leq \delta\right) \longrightarrow 1 \quad (n \longrightarrow \infty).$$

□

A similar result can also be proved for the convergence of the first derivative (see. Majidi, 2003).

Chapter 5

Extensions and Applications

Up to this point we restricted our examples to the case of artificial data sets with normally distributed noise where the procedures obviously work very well. Using two real data sets we now give two examples showing how to extend the method to obtain solutions for different problems arising in applications. The first are the Balloon data (see figure 5.1) which need a different analysis because they contain outliers. After describing a possibility to robustify the multiresolution conditions this new procedure is used to estimate the noise-level in non-homoscedastic cases. In section 5.3 we consider the thin-film data and solve the special problem how to approximate the baseline. This is done by combining the weighted smoothing spline with another nonparametric regression method, the taut string. Finally it is shown how the weighted smoothing splines can be used to obtain a smooth version of the result of the taut string method which has almost the same monotonicity as the former piecewise constant approximation and still satisfies the multiresolution constraints.

5.1 Robust Regression

Data sets obtained by real measurements often contain outliers, for example resulting from the way how the data have been collected. One example for such a data set is the Balloon data set (cf. Figure 5.1), which was first mentioned in Davies and Gather (1993).
The data consist of 4984 measurements of the radiation which were taken from a balloon about 30-40 kilometers above the earth's surface. The measuring instrument was suspended to the balloon by ropes which sometimes prevented the correct measurement of the direct radiation of the sun because due to the rotation of the balloon it was cut off by the ropes. Obviously it would be in vain trying to approximate the data by one of the described procedures. The multiresolution criteria constrain the approximation to reduce the bandwidth or increase the weights until the residuals look like white noise. In the Balloon data case this would only be possible if almost all of the outliers would have been adapted, the data would nearly be interpolated. Keeping the main idea of the procedures a different decision criterion is needed which takes the outliers into account and selects the smoothing parameters in a more robustified way.

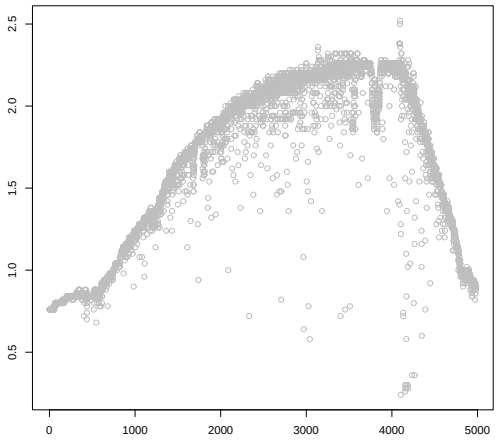


Figure 5.1: Balloon data.

To robustify the selection of local weights and bandwidths a robust version of the multiresolution criteria from Kovac (2003) can be used which relies on remarks of Dümbgen and Johns (2000) and Dümbgen (2001) .
Instead of the single residuals here the signs of the residuals are summed up to compute the MR coefficients:

$$w_{j,k} = \frac{1}{2^{j/2}} \sum_{i=k2^j+1}^{(k+1)2^j} \operatorname{sign}(r_i). \tag{5.1}$$

For $\varepsilon_1, \dots \varepsilon_n$ i.i.d. with $\mathbb{P}(\varepsilon > 0) = \mathbb{P}(\varepsilon < 0) = \frac{1}{2}$ the random variables

$$\frac{1}{2} \sum_{i=k2^j}^{(k+1)2^j-1} \operatorname{sign}(\varepsilon_i)$$

are asymptotically $b(2^j, 0.5)$ distributed. This leads to the idea to regard the residuals as noise if for all j, k:

$$|w_{j,k}| \leq \sqrt{2\log(n)}. \tag{5.2}$$

Hence in **Step 2** of the bandwidth or weight selection procedure now instead of (3.1) the robust criteria (5.2) are checked. Everything else of the method remains the same.
If a dataset contains some outliers the sensitivity of the local methods as for example the weighted splines is a drawback. In these cases the robust versions provide a better opportunity to obtain a satisfying result. Figure 5.1 shows the different behavior of the two spline approximation methods for a very simple and small example, a noisy sine containing a single outlier.

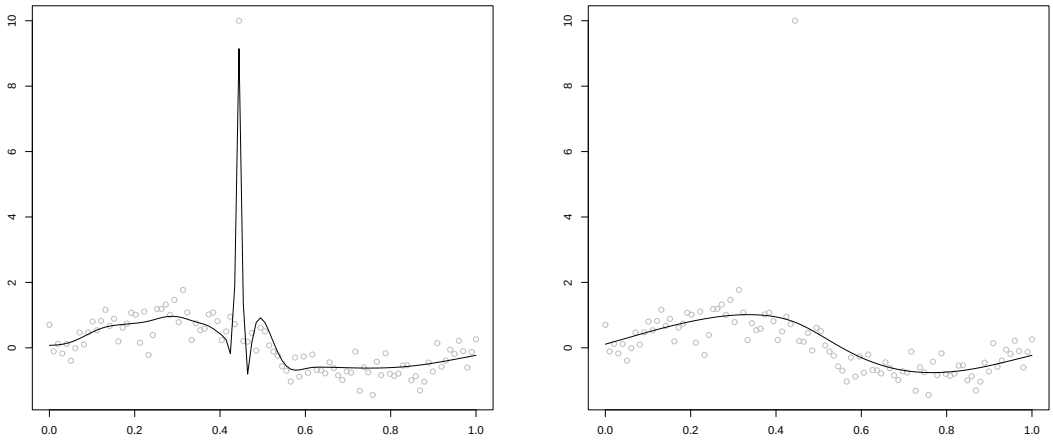


Figure 5.2: Comparison between the effect of one outlier in a noisy sine approximation for weighted splines with MR conditions (left) and its robust version.

However the procedure remains quite sensitive. Hence in the Balloon data case the weighted splines using the robust multiresolution criteria to specify the local weights obtain a devastating approximation, as it can be seen in figure 5.3. Since kernel estimators and splines are not robust at all, no satisfactory approximation can be obtained by using them for a data set which contains such a lot of and also large outliers as the Balloon data. But the robust multiresolution conditions can also be used for an automatically determination of the bandwidth for a robust regression technique. The easiest example for this is a local median which can be defined in the same way as the local mean in (1.3):

$$\hat{f}(t_j) = \text{Median}\{y_i, i \in I_j\}, \tag{5.3}$$

with $I_j = \{i : |t_j - t_i| \leq h_j\}$. Figure 5.3 shows the result, which in fact gives a reasonable approximation.

Apart from the direct approximation the robust method also provides an easy way to estimate the noise level for heteroscedastic data.

5.2 Heteroscedastic Data

Up to this point only homoscedastic data sets have been considered. The robust multiresolution criteria can now be used to estimate the noise level locally.
Starting from data (t_i, y_i) with

$$y_i = f_n(t_i) + \varepsilon_i, \tag{5.4}$$

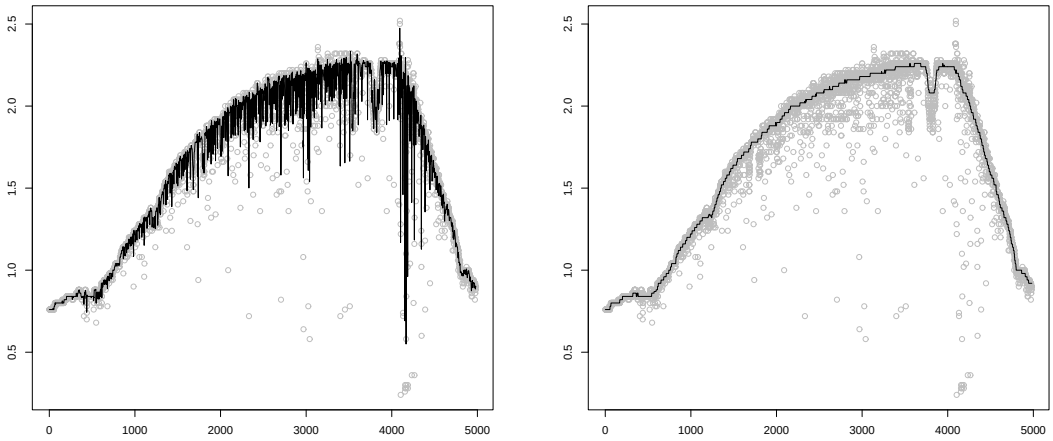


Figure 5.3: Left: Weighted spline approximation with locally chosen weights by using the robust multiresolution criteria. Right: Local median approximation with locally chosen bandwidth by using the robust multiresolution criteria.

and $\varepsilon_i \sim N(0, \sigma^2(t_i))$ distributed we obtain a first approach $\hat{\varepsilon}_i$ $i = 0, \ldots, n$ for the ε_i by the weighted smoothing spline using the robust multiresolution criteria. Therefore $g_n(t_i)$ $i = 0, \ldots, n$ is needed with

$$\hat{\varepsilon}_i^2 = g_n(t_i) \cdot r_i^2 \tag{5.5}$$

and r_i^2 i.i.d. $N(0,1)$, $g_n(t_i)$ is an estimation of $\sigma^2(t_i)$.
Again we are using an iterative procedure:

1. Computation of the $g_n(t_i)$ by a kernel regression of the $\hat{\varepsilon}_i^2$.

2. Checking, if the $r_i^2 = \dfrac{\hat{\varepsilon}_i^2}{g_n(t_i)}$ satisfy the criteria of the model.

3. Reducing the bandwidth for the kernel approximation if necessary.

In **1.** it is convenient to use a kernel regression, because in contrast to the smoothing splines it guarantees the positivity of the approximation. The quality test in **2.** relies on the known fact, that if Z_i, $i = 1, \ldots, n$ are random variables with Z_i i.i.d $N(0,1)$-distributed and $R_n = \sqrt{Z_1^2 + \cdots + Z_n^2}$ then $R_n^2 \sim \chi^2(n)$, i. e. R_n^2 is Chi-Square-distributed with n degrees of freedom.
In practice this means that in **2.** a two-sided test is applied:

$$C_1 \leq \sum_{i=k2^j}^{(k+1)2^j-1} r_i^2 \leq C_2 \tag{5.6}$$

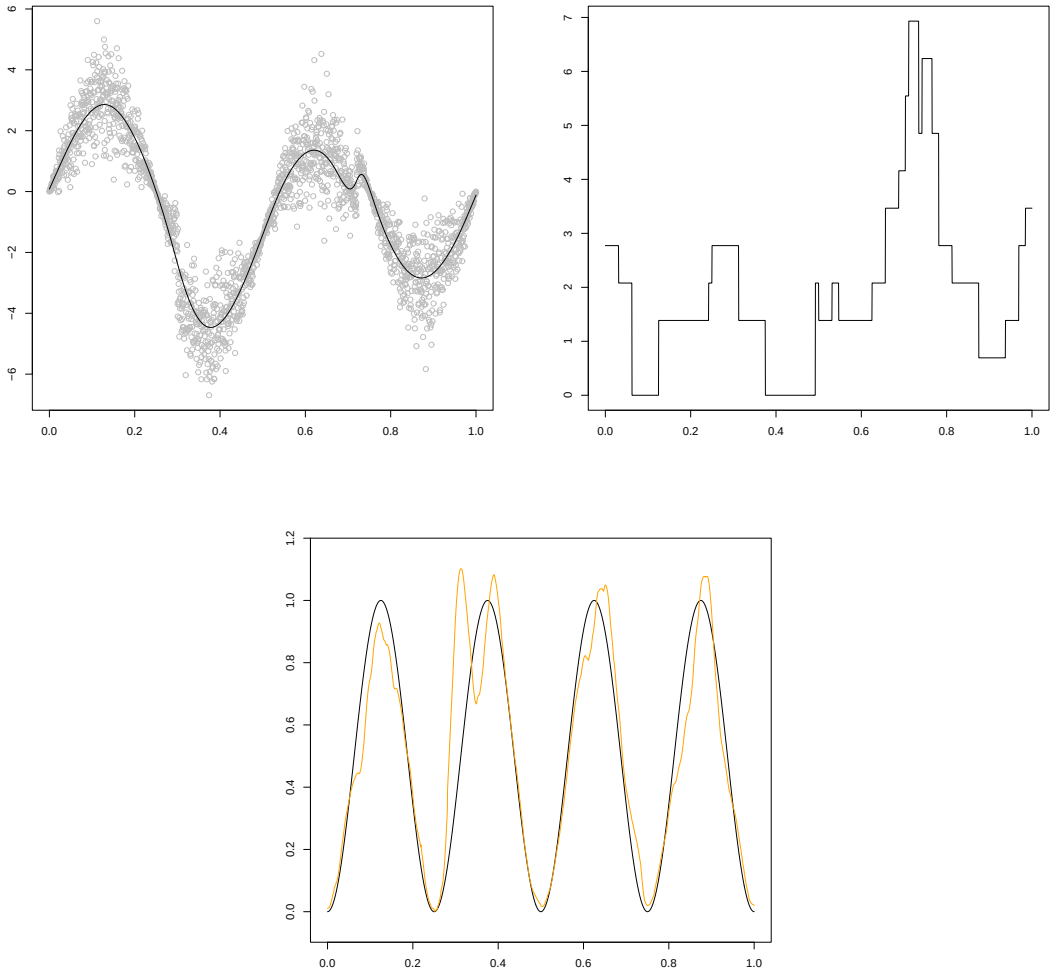


Figure 5.4: Robust approximation (top left) with locally adapted weights (top right, logarithmic) and the result of the noise level estimation (orange) with the true noise level (black).

with C_1: $\frac{0.5}{n}$-quantile and C_2: $(1 - \frac{0.5}{n})$-quantile of the Chi-Square distribution with 2^j degrees of freedom. The procedure is iterated until (5.6) is satisfied for all j, k.
Figure 5.4 shows the Heavisine function with added sine noise and the result of the approximation using the sum of signs. In the described way we obtain the approximation of the noise level which can also be seen in the mentioned figure.

5.3 Thin-Film Data

After the robustified procedures we now want to show what can be done to solve the special problems of the thin-film data. Because of their properties especially the weighted smoothing spline is very well suited for the described data set. As one would expect, it produces a very smooth baseline and also fits the peaks very well (cf. Figure 5.5).

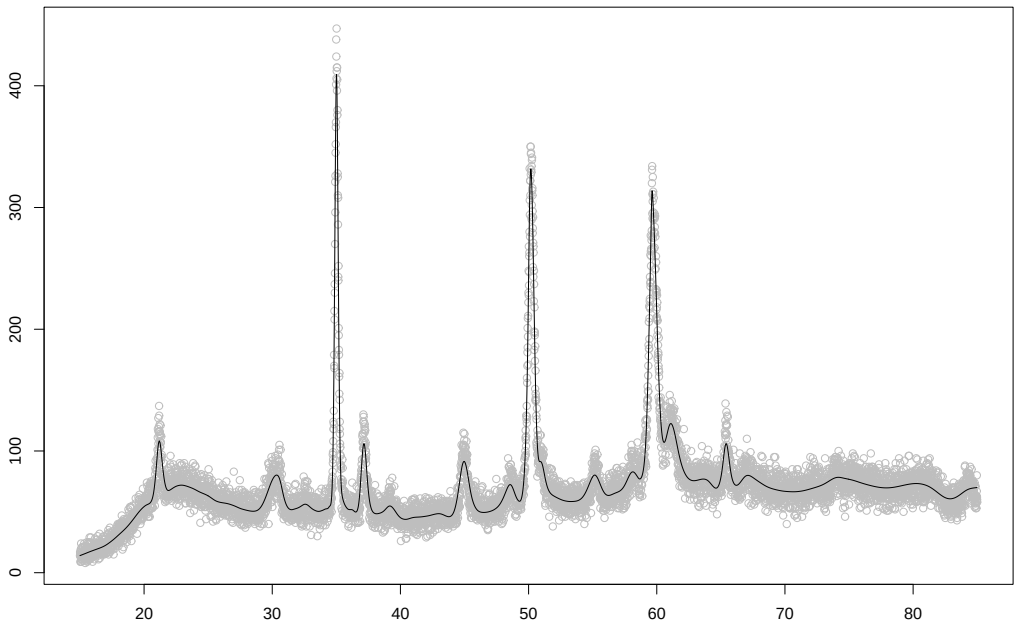


Figure 5.5: Weighted Spline approximation for the thin-film data.

One interesting problem in the context of the thin-film data is the baseline. Physicists are interested in height and width of the peaks but these are measured relatively to the baseline. That means for a better estimation of the peak properties it would be interesting to approximate the special trend of the baseline which depends on the measuring instrument. For example it can be seen in the data, that it is in the beginning slowly ascending but not remaining approximately constant in the further process. If the baseline is approximated it can be subtracted from the data and the peaks can be estimated with a special procedure.

We now want to propose one possibility how the weighted splines can be used to approximate the baseline of the thin-film data. To do so we want to combine the result of the splines with the approximation result of a different procedure, the taut string (see figure 5.6) from Davies and Kovac (2001).

Taut String

The approximation resulting of the taut string algorithm has special properties which can be easily understood following its idea and the way how it is computed. First the data $(t_i, y(t_i))$, $i = 0 \ldots, n$ with $t_i = \frac{i}{n}$ are integrated:

$$y_n^o\left(\frac{j}{n}\right) = \frac{1}{n}\sum_{i=0}^{j} y\left(\frac{i}{n}\right) \qquad j = 0, \ldots, n. \tag{5.7}$$

After specifying an upper and a lower bound $u_n = y_n^o + \frac{C}{\sqrt{n}}$ and $l_n = y_n^o - \frac{C}{\sqrt{n}}$ for some $C > 0$ at the left and right border a string is fixed in the middle of the resulting tube around the integrated data. This string is pulled tight so that finally it touches the tube at several points, the knots. The connection between these knots is given by a line. Differentiating the string returns the approximation of the data. The tube width can be chosen locally and is automatically selected in consideration of the multiresolution constraints. Because being the result of the differentiation of a piecewise linear string the approximation of the taut string is piecewise constant. Moreover it has the smallest number of local extreme values subject to some side conditions and it can be seen as a least squares minimization problem with total variation based penalty term:

$$\min_f \sum_{i=0}^{n}(y_i - f(t_i))^2 + \sum_{i=1}^{n}\lambda_i |f(t_i) - f(t_{i+1})|. \tag{5.8}$$

Because of its property of obtaining very few extreme values the taut string works very well in connection with the thin-film data. The approximation of the baseline parts is very smooth and the peaks are very well adapted. Because physicists are very pleased with the number of picked up peaks it was an easy decision to use this procedure and its special properties to obtain an approach to estimate the baseline. We start with the separation between baseline and peaks. Therefor we use both approximations of the thin-film data, the result of the weighted smoothing spline $\hat{f}_w$, and the result of the taut string $\hat{f}_{ts}$.

Baseline estimation

The taut string approximation $\hat{f}_{ts}$ provides the knot points of the local extreme values, that means for each local extreme value x_j we obtain an interval $I_j = [l_j, r_j)$, where $\hat{f}_{ts}(t)$ for all $t \in I_j$ is locally maximal or minimal. For an easier description we assume the equispaced t_i to be the indices just as l_j and r_j. $I = \{I_j\}$ contains all of the intervals where possibly peaks could be situated but the question remains how to decide automatically which real

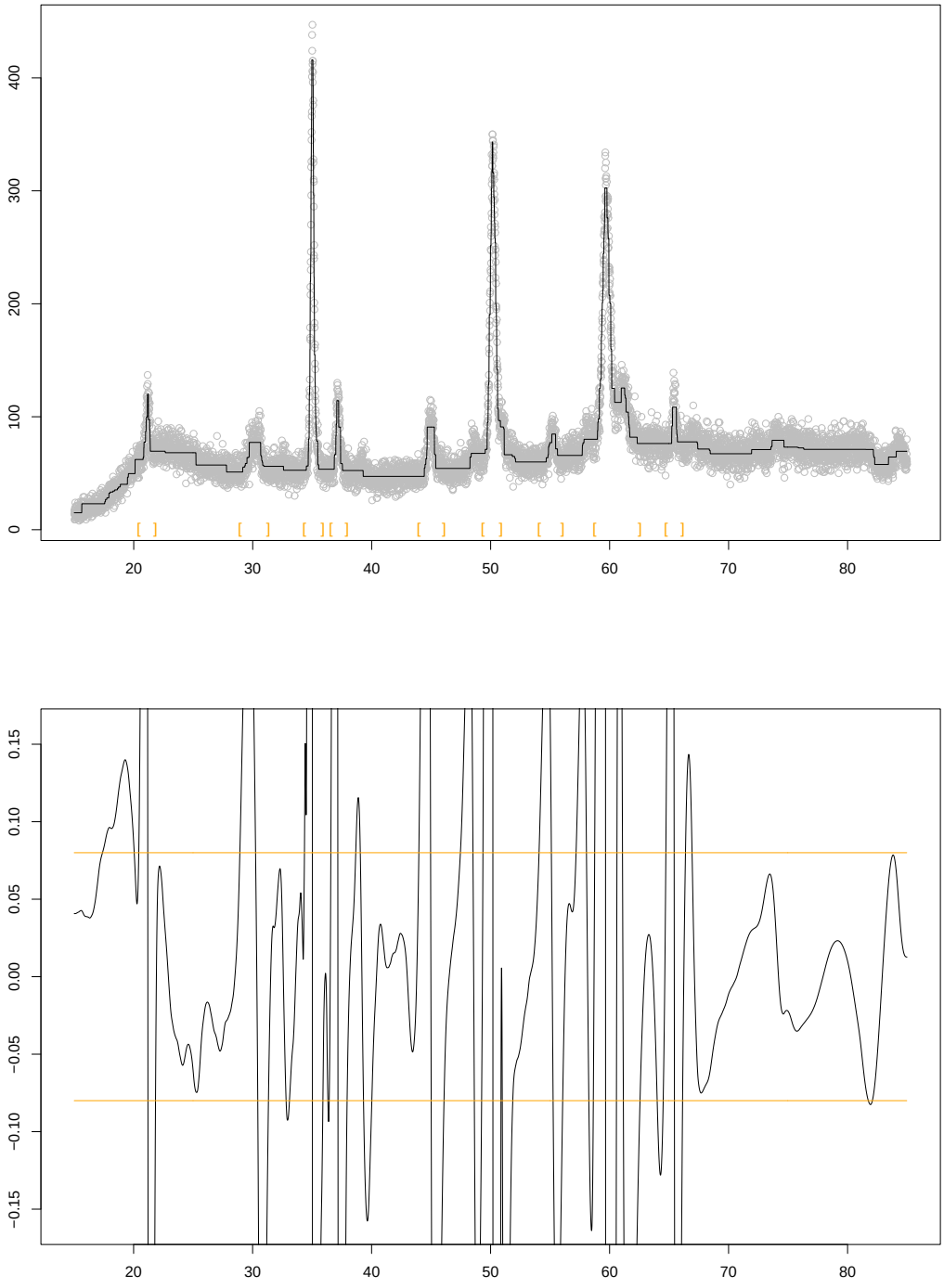


Figure 5.6: Top: Approximation of the taut string with marked peak intervals. Bottom: first order differences of the weighted spline approximation.

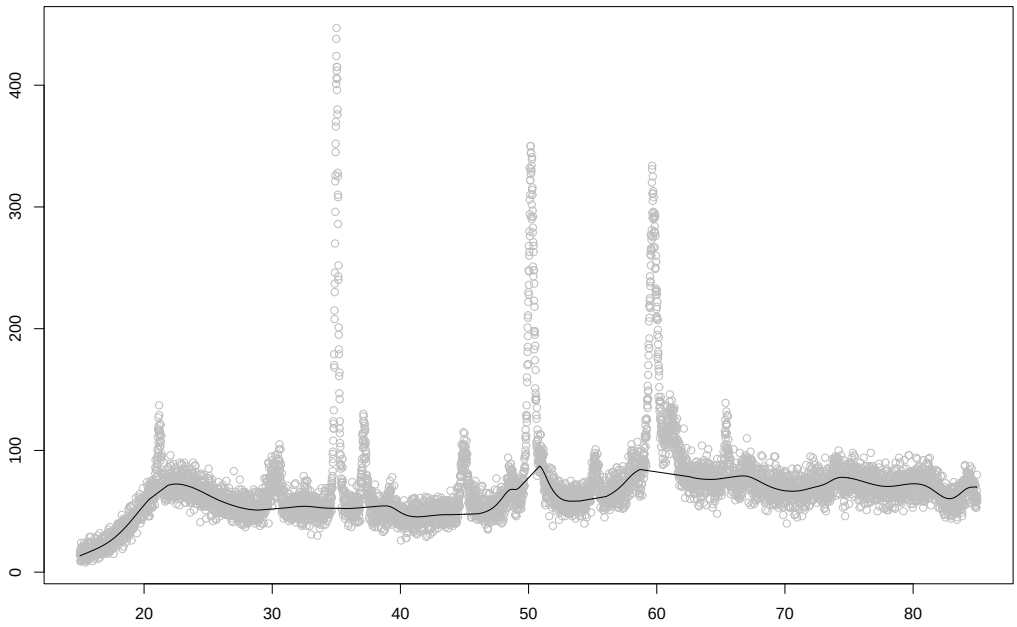


Figure 5.7: Baseline approximation using the weighted spline.

peaks and also how wide they are. First of all we have to determine which of the extreme values we count to the peaks and which we can neglect as belonging to the baseline.
Because of the special structure of the data we assume that all of the minima belong to the baseline except the minima which satisfy

$$\hat{f}_{ts}(t) > \text{Median}(y_0, \ldots, y_n) + MAD(y_0, \ldots, y_n) \qquad t \in I_j, \tag{5.9}$$

because they might result from the fact that two peaks are very closed together.
For the maxima we use the differentiability of the resulting spline and hence the first order differences of $\hat{f}_w$

$$d_j = \hat{f}_w(t_{j+1}) - \hat{f}_w(t_j) \qquad j = 0, \ldots, n-1, \tag{5.10}$$

to determine the widths of the peaks and also to decide if they are real peaks or not.
Therefore the interval $[l_j, r_j)$ is expanded until the smallest k and the smallest $\tilde{k}$ are found with

$$|d_{l_j-k+1}| - |d_{l_j-k}| \geq 0 \quad \text{and} \quad |d_{l_j-k}| \leq MAD(y_0, \ldots, y_n), \tag{5.11}$$

and

$$|d_{r_j+\tilde{k}}| - |d_{r_j+\tilde{k}+1}| \geq 0 \quad \text{and} \quad |d_{r_j+k}| \leq MAD(y_0, \ldots, y_n). \tag{5.12}$$

These constraints guarantee that the highest absolute slope has been passed and determine $\tilde{l}_j = l_j - k$ and $\tilde{r}_j = r_j + \tilde{k}$ where the peaks can be cut off. If

$$\max(|d_{\tilde{l}_j}|, \ldots, |d_{\tilde{r}_j}|) \leq MAD(y_0, \ldots, y_n) \tag{5.13}$$

then the local extreme value is not taken into account as a peak but determined as part of the baseline.
Figure 5.6 shows the approximation of the taut string (top) and the first order differences of the weighted spline approximation (bottom). The orange lines mark the boundary which is given by the MAD of the data. The picture at the top contains small orange brackets which determine the resulting intervals $[\tilde{l}_j, \tilde{r}_j]$ after the described determination process. These regions are cut off the data before estimating the baseline. This finally is done by the weighted spline using the same threshold as for the whole data set.
One possibility to approximate the baseline underlying the peaks would be to use the real spline interpolation but in this case it is sufficient to use a simple line connection. The final result of the baseline approximation can be seen in figure 5.7. The peaks itself can now be adapted separately.

5.4 Smoothing under Monotonicity Constraints?

One advantage of the taut string method as mentioned before in 5.3 is, that its result gives the information about extreme values and hence the monotonicity of a reasonable approximation. But it also obtains a piecewise constant approximation which in some situations is not satisfactory. The idea to find a smooth function which has the same monotonicity as the taut string and still fulfills the multiresolution conditions, turned out to be complicated. Majidi (2003) describes a method to solve approximately a discretized version of the problem on a regular grid. He minimizes the squared L_2-norm of the second derivative subject to the side conditions that the solution function has the same monotonicity as the given taut string approximation and that its integrated values lie in a tube around the data:

$$\begin{aligned} \|g''\|_2^2 = \min \quad \text{s. t.} \\ g \in W_2, \\ (g(t_{j+1}) - g(t_j))\mu_j \geq 0, \quad (j = 0, \ldots, n-1) \\ l_i \leq \mathcal{I}_n^k g(t_i) \leq u_i, \quad (i = 0, \ldots, n) \end{aligned} \tag{5.14}$$

with $W_m := \{g : [0,1] \to \mathbb{R} \,|\, g\, m\times\text{differentiable}, \int_0^1 (g^{(m)})(t)^2\, dt < \infty\}$, $\mu_j = \mu(t_j)$ the function which prescribes the monotonicity, $\mathcal{I}_n$ the discrete integration $\mathcal{I}_n(t_i) := \sum_{j=0}^{i} g(t_i)(t_j - t_{j-1})$ and l_i, u_i upper and lower bounds resulting from the multiresolution criteria. He proposes a combination of a multi-grid-QSOR and an active-set method. The two main problems of this idea are that the additional constraints of the multiresolution bounds destroy

the banded structure of the problem which means that the memory requirements can be very high. Moreover large differences in the second derivative cause numerical instability. Because of these problems the procedure is rather limited in its applications and therefore a more application-oriented method is of particular interest.
Hence we propose one possibility which combines the results of the taut string with the smoothness of a cubic spline and which obtains an approximation of the data satisfying the multiresolution conditions and has the same monotonicity except in a few points. The maximum number of these points can be specified in the following way. We allow the extreme values to be situated finally in some interval $[x_i - \delta, x_i + \delta]$, for x_i being the location of one of the given extreme values. That means that the points where finally the monotonicity can differ from the given one are restricted to these intervals.
We start with a processed version of the taut string approximation g_{ts}. Starting from the piecewise constant result of the taut string, the extreme values are fixed and then the parts in between are connected by piecewise linear functions in a way that the monotonicity remains the same and the multiresolution conditions are still satisfied. This piecewise linear approximation which can be seen in 5.8 still contains undesirable edges.

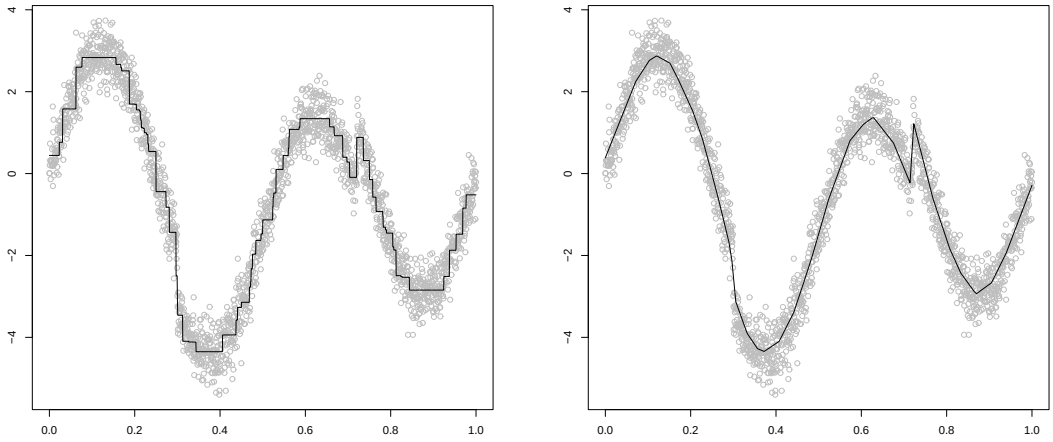


Figure 5.8: Taut string approximation(left) and smoothed taut string approximation (right) for the Heavisine data set.

Our idea now is, to compute a smoothing spline for a combination of the data y_i, $i = 0, \ldots, n$ and specific values of the smoothed taut string approximation $g_{ts}(t_i)$:

$$\tilde{y}(t_i) = \alpha_i y(t_i) + (1 - \alpha_i) g_{ts}(t_i) \tag{5.15}$$

Starting with the original data values $y(t_i)$, that means $\alpha_i = 0$ for all $i = 0, \ldots, n$, we replace the data values at the extreme value location with the extreme values of the smoothed taut string, $g_{ts}(t_i)$. More precisely, for the k extreme values of the smoothed taut string $(t_{i_j}, g(t_{i_j}))$, $j = 1, \ldots, k$ we set $\alpha_{i_1} = \cdots = \alpha_{i_k} = 0$. By doing so we want to force the smoothing spline to recover the fixed extreme values.

In the next step now the computation of the smoothing spline g_w for $\tilde{y}$ and the determination of the local weights go on as described in chapter 4 except that the residuals are computed by subtracting the real data y and not the manipulated ones $\tilde{y}$. When all of the local weights are determined in a way that g_w satisfies all of the multiresolution conditions the monotonicity adjustment begins. The iterative procedure is continued, however instead of controlling the multiresolution conditions during each iteration step now the extreme values of the approximated function are compared with the extreme values of g_{ts}. We distinguish between additional extreme points, missing ones and extreme values which correspond to one $(t_{i_j}, g_{ts}(t_{i_j}))$ of the given ones but are too far away from the location t_{i_j}, that means $|\tilde{t}_{i_j} - t_{i_j}| > \delta$ for a specified δ. In all of these cases the α_i are decreased by a factor of 0.5 and the weights are increased by a factor of 4 on the according intervals. These comparisons can be done easily because g_{ts} has a piecewise constant first derivative and g_w a continuous one. By doing so the influence of g_{ts} is increased stepwise in parts of the data set where g_w does not satisfy the monotonicity conditions. At the moment when the number of the extreme values of g_w is equal to the number of the given extreme values of g_{ts} and also their locations are close enough, again the multiresolution conditions are checked and if necessary some weights are increased.

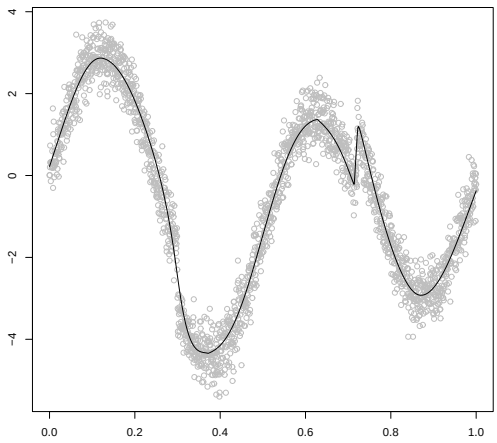


Figure 5.9: The resulting spline approximation with the same monotonicity as the given taut string.

So that finally the new approximation (see e. g. Figure 5.9) has almost the same monotonicity and is in most of the cases smoother than g_{ts} (in the worst case g_w is almost similar to g_{ts}) because of the spline properties and still fulfills the multiresolution conditions. The procedure shows that the resulting approximation can still contain some features of the smoothed taut string, but it provides a simple possibility to avoid the piecewise constant approximation. Moreover this method obtains a better approximation of discontinuities like the second one of the Heavisine data set as the simple local weighted smoothing spline.

Chapter 6

Bivariate Smoothing

After providing several procedures for local smoothing in one dimension we now want to draw the attention to some aspects of similar bivariate problems.
In the context of smoothing in two dimensions we can distinguish between two aims of interest and application. The first one can be described as the denoising of images. In this context often piecewise constant regions are desirable, as for example in different medical applications, and it is important to recover the edges of the different features contained in an image. The second one draws the attention to continuous approximations of the data and can be seen as the fitting of a surface which has certain properties. These properties depend on the context of the application.
Since different forms of splines are very successful in univariate situations it is evident to use the minimization of penalized (in-)fidelity terms also in the bivariate context. The choice of fidelity and penalty term determine the properties of the resulting approximation. Hence these terms have to be chosen in such a way that the desired application is taken into account.
As one aspect of particular interest we want to concentrate on data sets with a very large noise level or large outliers and hence we now present a procedure which can be seen as an extension of the robust version of the taut string method (cf. section 5.1 and 5.3). An L_1-fidelity term is used, penalized by a total variation based term which measures the roughness of the approximation. In this context again a locally defined smoothing parameter is determined automatically, now by a two dimensional version of the multiresolution criteria. Thereby we obtain the possibility to influence the approximation locally. As mentioned above, this method is a robust procedure so that it can be applied to noisy data with very large outliers, where other bivariate spline methods fail because of their sensitivity.

6.1 Idea

The idea of the method which will be described in the sequel is to use the basic idea of the multiresolution to denoise images and especially noisy images with very large outliers. In section 5.1 we introduced the robust multiresolution conditions which are used by Kovac

(2003) to determine the local smoothing parameters in a robust version of the taut string. This robust version of the taut string gives an approximation in the one dimensional case which satisfies both of the indirectly mentioned properties of such a regression method we are looking for. It is a robust procedure and it adapts a piecewise constant function which seems to be very useful for image denoising. Hence the idea is to extend this method to the bivariate case, where we have data $(t_{i,j}, y(t_{i,j}))$, $i, j \in \{0, \ldots, n\}$, $t_{i,j}$ on an equispaced grid ξ_{n^2}. For a convenient description we assume that the grid ξ_{n^2} is a grid on $[0,1] \times [0,1]$. That means, we have grid points $t_{i,j} = (\frac{i}{n}, \frac{j}{n})$, $i, j \in \{0, \ldots n\}$.
In the one dimensional case the approximation of data $(t_i, y(t_i))$, $i = 0, \ldots, n$ obtained by the robust taut string can be seen as the solution of the following minimization problem:

$$\min_f \sum_{i=0}^{n} |y(t_i) - f(t_i)| + \sum_{i=1}^{n} \lambda_i |f(t_i) - f(t_{i-1})|. \tag{6.1}$$

This notation shows that the robust taut string is an L_1-approximation of the data penalized by the total variation of the approximated function. Therefore an extension to two dimensions leads to a penalized L_1-minimization problem.
Functions of bounded variation are very important in the context of image denoising and during the last years a lot of work has been done in this area. Rudin, Osher and Fatemi (1992) introduced a very efficient regularization method for images following the objective to recover also sharp edges. Using that the total variation of a differentiable function on $\mathbb{R}^2$ can be seen as

$$\int |\nabla f| \tag{6.2}$$

they minimize

$$\int \sqrt{f_x^2 + f_y^2} \tag{6.3}$$

subject to constraints involving the mean and the standard deviation of the noisy image. The procedure gives very good results and allows a fast computation. For example Chambolle and Lions (1997) and Chambolle (2003) show the link between the constraint minimization problem and the minimization of the associated Lagrangian functional

$$\min_f \frac{1}{2\lambda} \|y - f\|_2^2 + \int |\nabla f| \tag{6.4}$$

and a discretized version. A very fast and easy algorithm is provided which computes the solution for an estimated σ. In this procedure both the fidelity and the penalty term lead to a differentiable approximation which still recovers the edges of the included features of an image very well.
But considering the L_1-minimization we now replace the L_2-norm in 6.3 by the L_1-norm and use a discretized approximation. To obtain enough flexibility although we are interested

in the robustness of the procedure we introduce a locally defined smoothing parameter $\lambda_{i,j}$. Hence the minimization problem corresponding to (6.1) becomes

$$\min_f \sum_{i,j=0}^{n} |y(t_{i,j}) - f(t_{i,j})| + \sum_{i=1}^{n} \lambda_{i,j} |f(t_{i,j}) - f(t_{i-1,j})| + \sum_{j=1}^{n} \lambda_{i,j} |f(t_{i,j}) - f(t_{i,j-1})| \quad (6.5)$$

As mentioned before, a two-dimensional and robust version of the multiresolution criteria guarantees a satisfactory data closeness by determining the smoothing parameter automatically. Additionally, we want the resulting piecewise constant regions of the approximation to be as large as possible. Therefore an iterative procedure is used to determine the smoothing parameters. To avoid artefacts the iteration starts with large values $\lambda_{i,j}$ and reduces them stepwise.
Another approach in the context of two dimensional L_1 regression are the penalized triograms from Koenker and Mizera (2004) which will be mentioned in chapter 7. Here the total variation of the first derivative is used as roughness penalty term. Moreover the paper contains an extensive reflection about total variation in general and especially for bivariate functions. In this context total variation is not as clear as in the one dimensional case.

6.2 Procedure

After the short overview about the general idea of this method we now want to describe its realization. Again we use an iterative procedure, which combines the regression for fixed smoothing parameters $\lambda_{i,j}$ with a control step in which the goodness-of-fit of the approximation is checked locally:

Step 1 computing the solution of (6.5) for given $\lambda_{i,j}$,
Step 2 checking the multiresolution conditions,
Step 3 reducing the local smoothing parameters where necessary.

6.2.1 Linear Programming

To solve the minimization problem of **Step 1** standard procedures of linear programming can be used. Koenker and Portnoy (1997) for example show how L_1-minimization problems can be translated into the standard form of linear programming. Using the same procedure we translated (6.5) to the form needed in the primal-dual interior-point method of Mehrotra (see e. g. Nocedal and Wright, 1999, or Wright, 1994 and in the context of L_1-regression Koenker and Portnoy, 1997). Additionally, the sparsity of the design matrix can be exploited (cf. Koenker and Ng, 2002). But although the design matrix is sparse, the complexity of the problem can become very large. For example it is not guaranteed that the Cholesky decomposition of a sparse matrix again has only few entries. Resulting from the locally defined parameter $\lambda_{i,j}$ and the high dimensionality the procedure can become numerically unstable. Hence it is very important to use some elaborate and fast software. Besides an own implementation we use the software provided by the `nprq` package for

R from Koenker and Ng, which we also used to create the examples in section 6.3. But still there are several aspects of programming which could be exploited to improve the applicability of the method.

6.2.2 Multiresolution in Two Dimensions

After solving the minimization problem for fixed values $\lambda_{i,j}$ in **Step 2** the regions need to be determined in which the approximation is not close enough to the data and hence the $\lambda_{i,j}$ have to be decreased. Following the idea of the described one dimensional procedures we are looking for a two dimensional equivalent of the multiresolution check. Referring to chapter 2 it can be said that an adequate approximation would satisfy the following inequalities for all subsets I of $[0,1]^2$:

$$\frac{1}{\sqrt{N_I}}|\sum_{(t_i,t_j)\in I} r_{i,j}| \leq \sigma_n\sqrt{2\tau\log(n)} \tag{6.6}$$

with $r_{i,j} = y_{i,j} - \hat{f}_{i,j}$, $i,j = 0,\ldots,n$ the residuals, $\hat{f}_{i,j}$ the solution of (6.5) and N_I the number of grid points in I. According to (3.4) σ_n is approximated by

$$\sigma_n = \frac{1.48}{2}\text{Median}\left\{|y_{i+1,j+1} - y_{i+1,j} - y_{i,j+1} + y_{i,j}|,\, i,j = 0,\ldots,n-1\right\} \tag{6.7}$$

Obviously, it can be very time extensive to check this criterion on all partitions of the square. In the one dimensional case we reduce the requirement to the dyadic intervals. But in two dimensions this reduction is not as easy. The choice of the subsets determines the regions where the smoothing parameters will be reduced afterwards, hence too large limitations can influence the edges of the image. For example it can be said that the equivalent to the dyadic intervals in the one dimensional case are dyadic squares. Assuming $n+1 = 2^l$ that means the square $[0,1]^2$ is divided into four equal squares, then each of the four squares is again divided into four squares and so on. Obviously this is a very strong restriction but also a computationally very fast possibility. Often it will be satisfactory just to check all of these dyadic squares but there are also cases where this is not sensitive enough and the edges of the features would have a quadratic appearance. Polzehl and Spokoiny (2003) concentrate on the question of how to choose areas in equidistant designs and offer a kind of multiresolution criterion where squares and their linear divisions are checked. But they also remark that the computation is very extensive and hence not really applicable. Therefore for our examples we use a compromise by just considering all partial squares of $[0,1]^2$. But we again suggest that the use of dyadic squares sometimes can be sufficient and of course speeds up the procedure and that the additional check of linear splits of the squares, e. g. triangles can improve the recovering of the edges. For the multiresolution hence we check if all squares Q_k of $[0,1]^2$ with side length containing $k = 1,\ldots,n+1$ grid points satisfy (6.6) with $I = Q_k$.

Robust Multiresolution

As mentioned before, the approximation characterized by the L_1-fidelity and the total variation penalty term is very appropriate for robust denoising. To tap this potential, the procedure can be combined with special robust multiresolution constraints. In section 5.1 the multiresolution sum of signs have been considered. Here we want to work with truncated residuals, using $3\sigma_n$ as threshold. That means the residuals $r_{i,j}$ in (6.6) are replaced by

$$\tilde{r}_{i,j} = r_{i,j}\mathbf{1}_{\{|r_{i,j}|\leq 3\sigma_n\}} + \operatorname{sign}(r_{i,j})\, 3\,\sigma_n \left(1 - \mathbf{1}_{\{|r_{i,j}|\leq 3\sigma_n\}}\right), \tag{6.8}$$

$\mathbf{1}_M$ being the indicator function of a set M. In doing so it is possible to recover also images with very large noise as it can be seen in section 6.3.2. Another context would possibly need a different threshold, here $3\sigma_n$ is very suitable.

6.3 Results

To present the results of the described smoothing method we use different artificial data sets. One very simple image which contains only three different features, an ellipsoid, an annulus and a straight line, is shown in Figure 6.1. These three shapes provide various problems as curve edges, very small features and a thin straight object. Considering this example we want to demonstrate the advantages and also difficulties of the procedure in the context of image denoising. The second example can be seen in Figure 6.2. It is the graph of the function $z(x, y) = 4-4x^2+4y^3$ on $[-1, 1]^2$, here an equispaced grid of 128×128 is used. Both functions are used with added normal noise and also with Cauchy noise.

6.3.1 Normal Distributed Noise

Images

Figure 6.1 shows the first data set adding normally distributed noise. A satisfactory procedure would remove the noise and reproduce the different constant regions with sharp edges. The result of the L_1-method with total variation penalty can be seen in Figure 6.1. The figure on the left hand side of the bottom shows the result with satisfied multiresolution constraints (**Step 2**) on each of the squares Q_k. Using the multiresolution conditions only on dyadic squares in this simple case does not make much difference, hence we omit an illustration. The approximation features the desired property of large constant regions. Also the edges are recovered relatively well, but not in all parts as good as one would wish. This and also the problems with very small features, as for example the little circle in the annulus, or the thin straight line are due to the robustness of the L_1-minimization.
The properties of the result become more obvious in a cut through the data and its approximations. Figure 6.1 shows the 65th column of the data. The black line marks the corresponding column of the original image and the orange one the one of the approximation. This shows that the method is able to approximate large constant areas as for example in the right part of the figure. Also the edges are fitted very well and the curvature

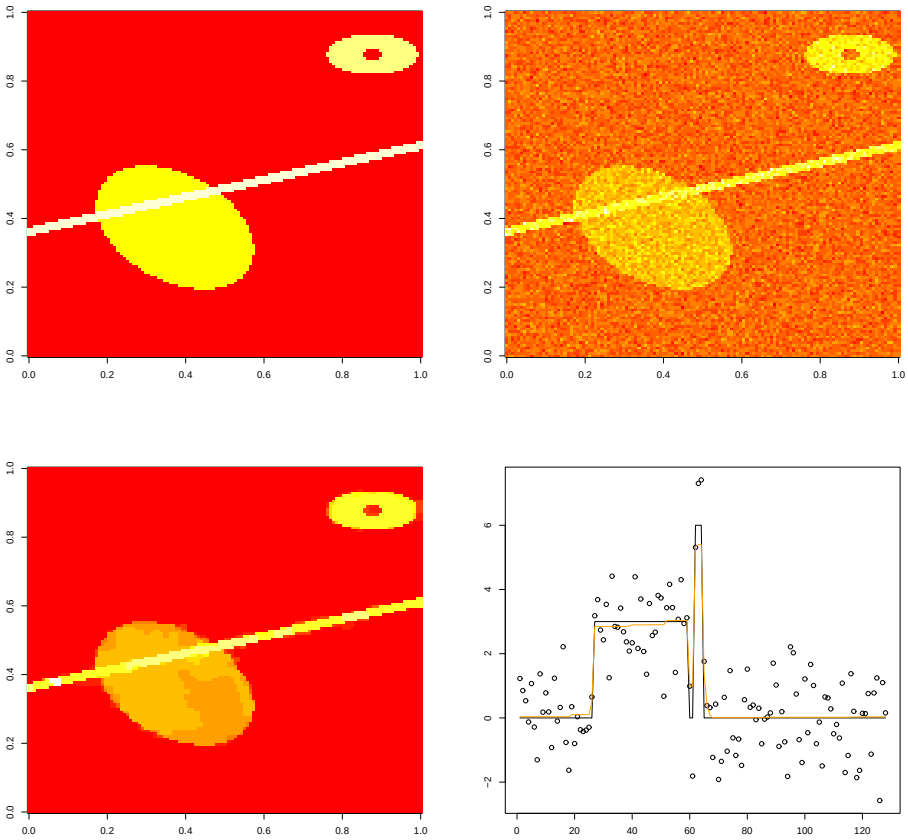


Figure 6.1: Top: Test image, original (left) and with added normally distributed noise (right). Bottom left: Approximation using the multiresolution check on all squares. Bottom right: A cut through the approximation. Black line: original image, orange: approximated one.

is recovered. The underestimation of maxima and the overestimation of the minima, here seen as function in one dimension, result from the properties of the procedure. Compared to the check on the dyadic squares, the multiresolution check on all squares improves the fit. It becomes more sensitive and hence little features are recovered in a better way. But this also provides the risk to be too sensitive and to have artefacts in the approximation, like additional extreme values, which are very rare otherwise.

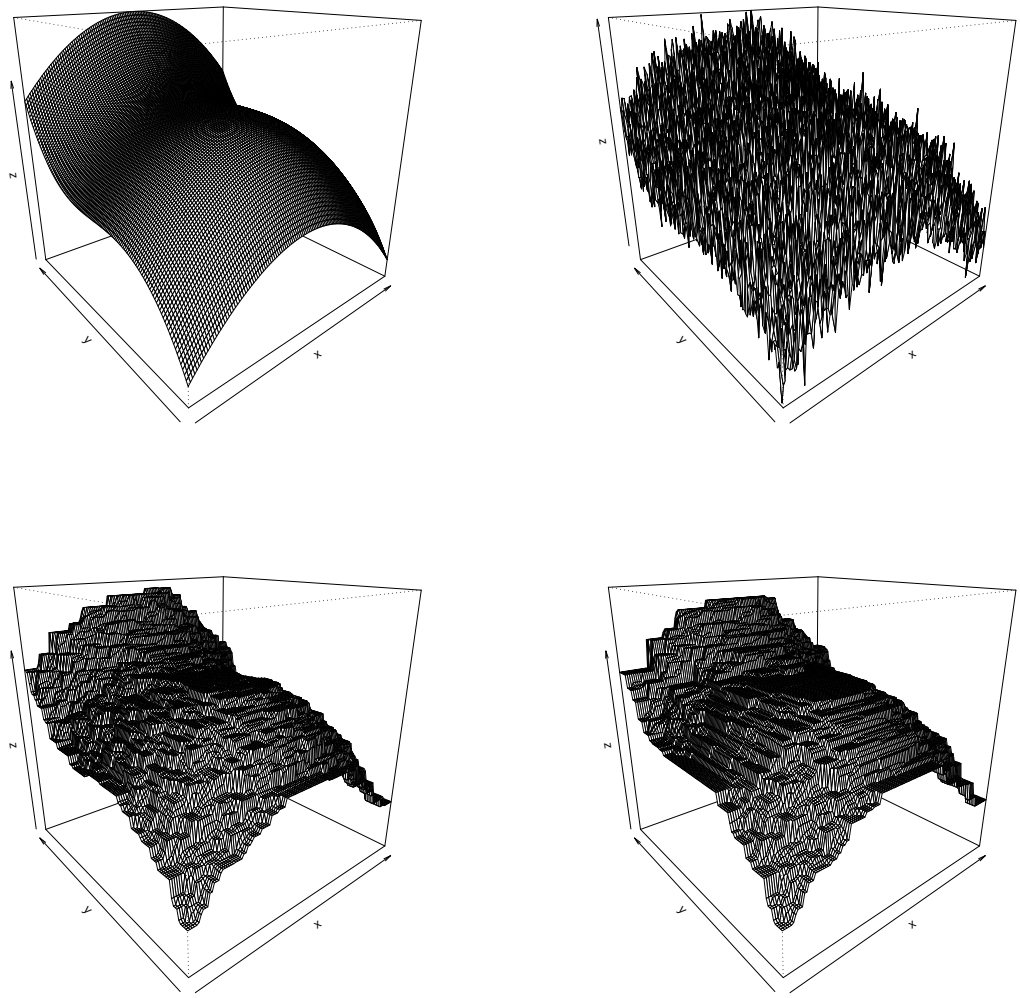


Figure 6.2: Top: The original function (left) and with added noise (right). Bottom: Approximation of the noisy saddle function. Left: using the multiresolution criteria on all partial squares, right: multiresolution only on the dyadic squares.

Surfaces

Looking at the second example the property of recovering the curvature becomes more clear. In Figure 6.2 the noisy saddle function can be seen. The Approximation in Figure 6.2 shows the piecewise constant approximation. Even though the resulting approximation is not smooth, the curvature of the original function is almost recovered. Of particular importance is the lack of additional maxima, no artefacts arise. Here now both the approximation using the multiresolution criteria only on dyadic squares and on all squares Q_k can be seen. Obviously the second version shows the above mentioned improved sensitivity.

6.3.2 Large Noise

Images

In the next examples the multiresolution check in **Step 2** is computed using truncated residuals. To show the results of this robust version, the same images have been used with added Cauchy noise. Hence the outliers in the data are very large, therefore the original image is not visible any more in the default plot function of the Statistics Software package R, which can be seen in Figure 6.3.2. This figure also shows the computed approximation using the described procedure. Compared with the result for the normal noise, obviously the features are not recovered as good as before, but looking again at a single column of the data and its approximation (Figure 6.3.2) the difficulties become clear. The picture on the left hand side shows the size of the outliers. The same column plotted on a different scale on the right hand side reveals the goodness of the approximation. Obviously little features are not fitted very well, as for example the small gap between the ellipsoid and the line and also the line itself. But this is what has to be accepted as side effect of the improved robustness of the procedure. The function values are relatively small compared to the noise level. Using the same image with increased function values on the three features the procedure gives a better result. (Figure 6.4).

Surfaces

The result of the procedure for the saddle function with added Cauchy noise is also satisfactory (Figure 6.4). Again the multiresolution conditions are computed using truncated residuals. And obviously the procedure still fits a function which recovers the curvature very well. Sometimes artefacts occur resulting from the large noise level. One of those can be seen in the figure of the approximation using multiresolution conditions on all partial squares. It results from the fact that in this case several very large outliers are very close together so that the direction is fitted but none of the outliers is really adapted. As mentioned before, the increased number of multiresolution conditions makes the procedure more sensitive, hence only this version shows the artefact.

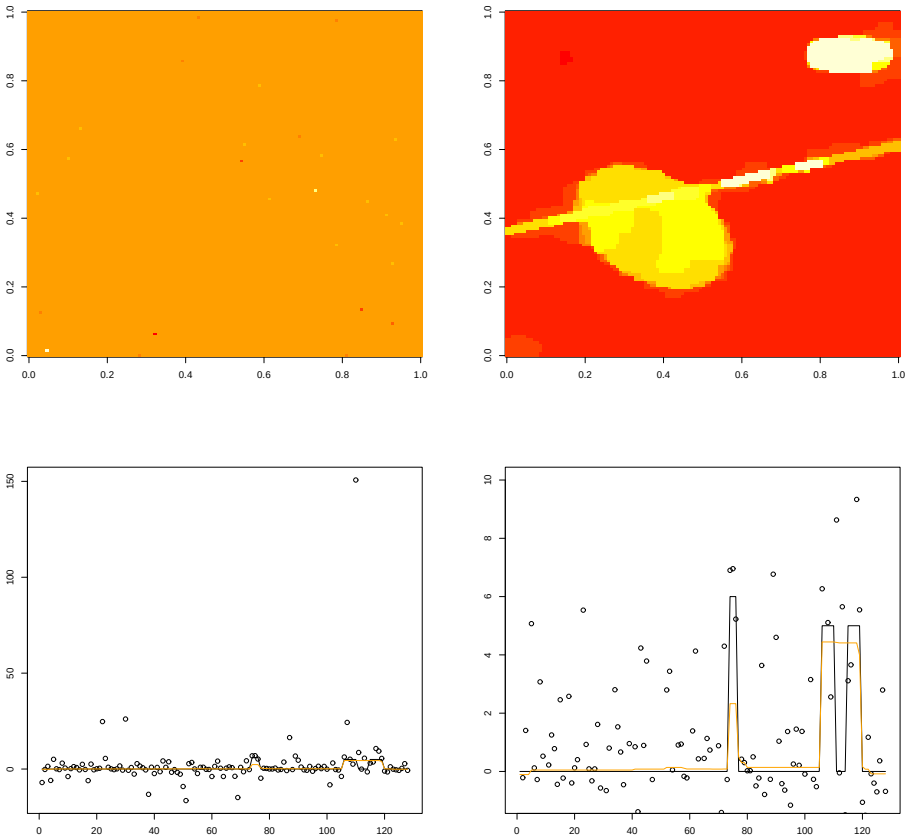


Figure 6.3: Top: The original image of Figure 6.1 with added Cauchy noise (left) and the result of the smoothing method (right) using truncated residuals for the multiresolution conditions.Bottom: One column of data, original function (black) and approximation (orange) on two different scales.

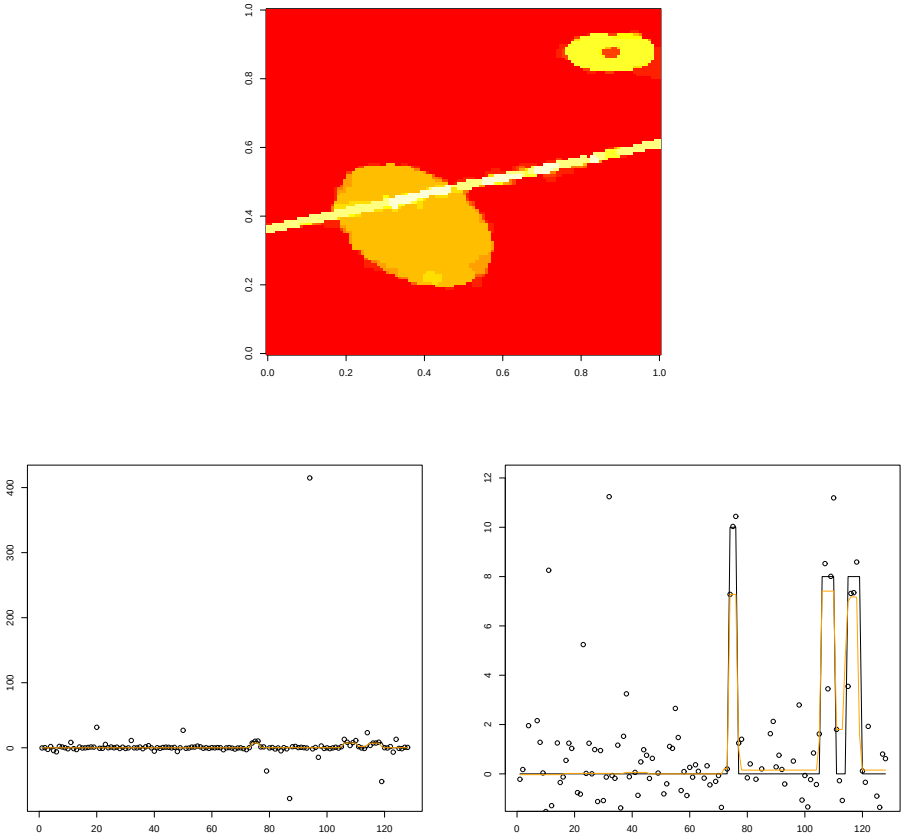


Figure 6.4: Top: approximation of the image with larger function values and added Cauchy noise. Bottom: one single column plotted on different scales. Black line: the original function, orange: the approximation.

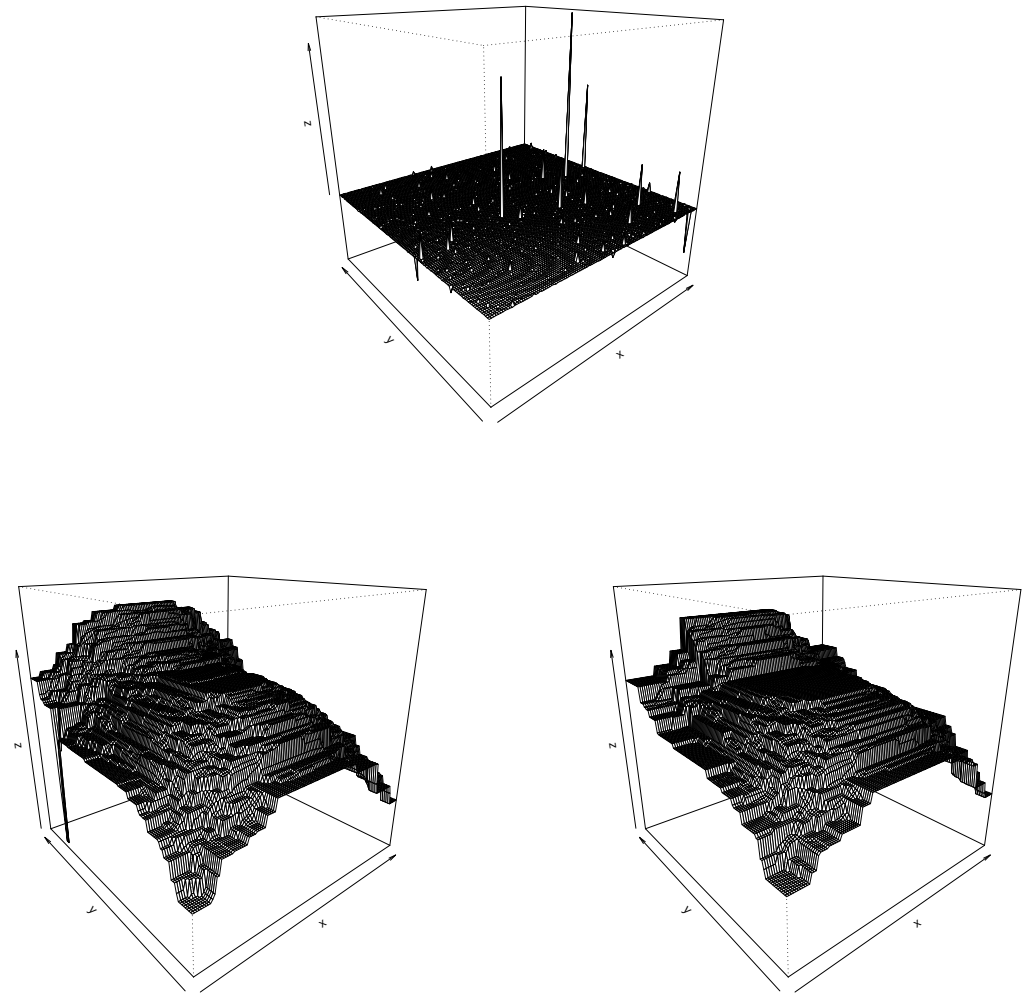


Figure 6.5: Top: Saddle function with added Cauchy noise. Bottom left: result of the method using robust multiresolution conditions on all partial squares. Bottom right: result of the method using robust multiresolution conditions on dyadic squares

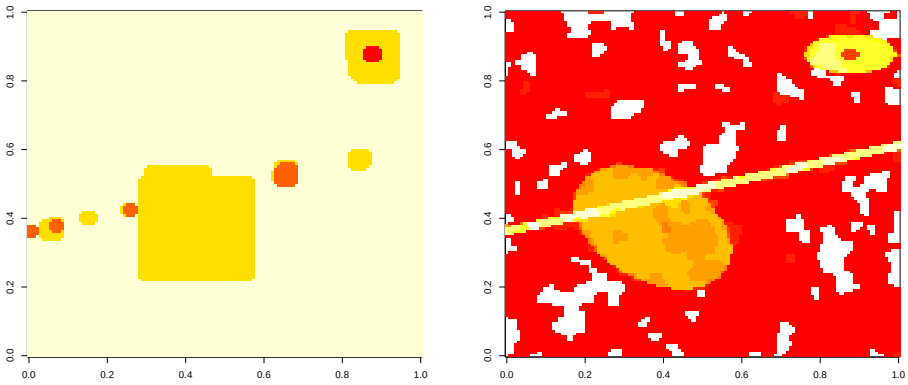


Figure 6.6: Left: The locally adapted smoothing parameter resulting from the approximation of Figure 6.1. Right: The same approximation as in Figure 6.1, but using a globally defined smoothing parameter.

6.3.3 Smoothing Parameter

Even though the examples are relatively simple, the result of the automatically adapted smoothing parameter shows its importance. In Figure 6.6 the smoothing parameters of the approximation which is shown in Figure 6.1 can be seen. Obviously small features as the little circle in the inside of the annulus need a smaller value of the smoothing parameter to be approximated well enough. Also the ellipsoid and the line left their marks in the values of the smoothing parameters. Without the local version of the smoothing parameter the rest of the image would not be as smooth and the constant regions would not be as large as they do now. The result of the same procedure but using a globally defined smoothing parameter is shown on the right hand side of Figure 6.6. The approximation which satisfies all of the multiresolution conditions gives a good recovering of the edges of the three features but on the other hand it does not have as large constant regions as the result of the local version. To visualize the difference, the image is plotted on the same scale as the result of the local version in Figure 6.1.

6.4 Conclusion

Concluding we can say that also the bivariate smoothing procedure with the automatically determined smoothing parameter gives results with very satisfactory properties. The robustness of the method and hence the very small number of artefacts is an advantage as well in the normal noise examples as in the ones with Cauchy noise. Discontinuities or non differentiable parts can be approximated very well. However, as a consequence the approximation is not smooth but piecewise constant. The curvature is very well recovered. Critically we have to point out some other aspects of the method especially in the context of denoising normal noise images. Leading to a piecewise constant approximation, the procedure seems to be suitable for image denoising. But a satisfactory image denoising

technique needs to be very sensitive. The idea was to use a locally defined smoothing parameter to reach this necessary sensitivity. But computing the multiresolution conditions on more than the used squares increases the complexity so that an application would not be possible. Anyway the computational effort is very large, hence only small images can be treated (the example pictures have size 128×128). Here special algorithms would be necessary to improve computing time and memory requirements. With an increased number of data points a better adaptivity of the approximation is obtained. Although there are

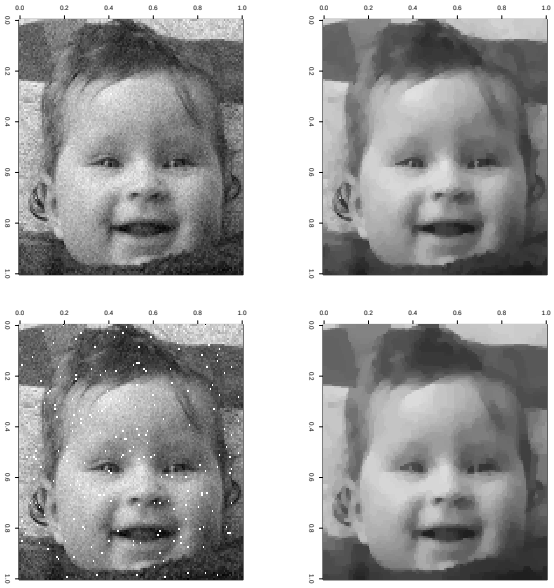


Figure 6.7: Top: A picture with added normal noise (left) and its approximation (right). Bottom: The same picture with 200 additional outliers (left) and its approximation (right).

several things which could be improved, the automatic determination of the locally defined smoothing parameter provides a good possibility to approximate also little features without influencing the surrounding area. And the combination of the smoothing procedure with the robust multiresolution conditions provides a method to denoise also images with very large noise level. Here not only the curvature can be recovered very well but also relatively small features become visible. In this context too high sensitivity would be a drawback. Figure 6.7 shows this balance between robustness and sensitivity. The picture on the bottom (left) contains beside the added normally distributed noise 200 outliers as they can result from dirt or little scratches. The values in these points are more than ten times larger then the real picture values. The approximation does not differ very much from the approximation of the normal noisy picture but obviously this procedure is not sensitive enough to satisfy a photographer who is interested in a very detailed picture. Hence this procedure is more suitable for images with larger features.

Chapter 7

Global Smoothing Parameter and Asymptotics

Certainly it is not only possible to use the multiresolution criteria to determine locally defined smoothing parameters but also to determine globally defined ones as for example those used to control the influence of the roughness penalty in the case of smoothing splines. In the one dimensional case the determination of a global parameter is not of large interest because localized versions of the standard procedures are easy to obtain and do not increase the computing time too much. Compared to this, the two dimensional situation seems to be different. The computation of bivariate spline versions is mostly time and memory extensive and the special algorithms do not allow an easy localization. Therefore we want to give two examples for an application of the multiresolution criteria to determine a global smoothing parameter. The thin plate splines as natural bivariate extension of the penalized least squares approximation of the univariate splines were proposed by Duchon (1975, 1976, 1977) and Meinguet (1979). For example Wahba (1990) and Green and Silverman (1994) provide good overviews about the topic and interesting references. Thin plate splines fit a very smooth and twice differentiable surface to the data whereas the penalized triogram approximation (Koenker and Mizera, 2004), using an L_1-fidelity and a total variation based penalty term, fits a piecewise linear function and hence improves the approximation at rough edges.

To demonstrate the results we use an artifical data set, a noisy cone

$$z_i = \max\left\{0,\ \frac{1}{3} - \frac{1}{2}\sqrt{x_i^2 + y_i^2}\right\} + \varepsilon_i \quad , i = 0, \ldots, N = (n+1)^2 - 1 \tag{7.1}$$

with (x_i, y_i) equispaced grid points on $[-1, 1]^2$ and the ε_i i.i.d. Gaussian noise with $\sigma = 0.02$. Figure 7.1 shows the original function and the noisy data using $N = 200$.

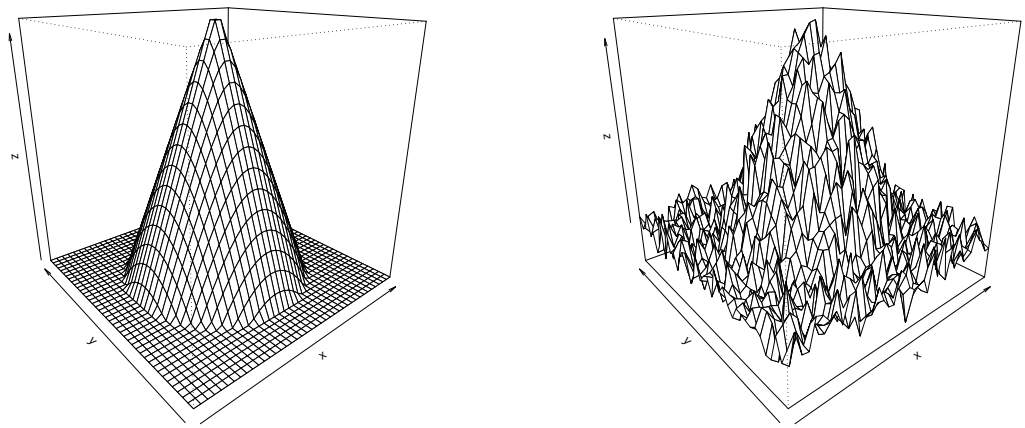


Figure 7.1: The original function (left) and the noisy cone (right).

7.1 Thin Plate Smoothing Splines

The idea which leads to the thin plate smoothing splines is to extend the penalized least squares idea of univariate smoothing splines to regression in higher dimensions and especially to bivariate situations. As described in chapter 4, the penalty term of the smoothing splines, the integral of the squared second derivative, measures the roughness of the fitted curve. Hence the question for the extension of the method was, how to measure the roughness of a surface. Green and Silverman (1994) describe which desirable properties of such a roughness functional, as for example the invariance in respect to rotation or translation, finally resulted in the choice of

$$J(g) = \int \int_{\mathbb{R}^2} (g_{xx}^2 + 2g_{xy}^2 + g_{yy}^2)\, dx\, dy. \tag{7.2}$$

This thin plate penalty term can be seen as a natural bivariate extension of the univariate roughness penalty, as for example shown in Koenker and Mizera (2004). The corresponding thin plate smoothing spline is then given by the minimizer of:

$$\min_g \sum_{i=0}^{n} (z_i - g(x_i, y_i))^2 + \lambda J(g). \tag{7.3}$$

Green and Silverman (1994) provide the finite window thin plate splines, using

$$J_\Omega(g) = \int \int_{\Omega} (g_{xx}^2 + 2g_{xy}^2 + g_{yy}^2)\, dx\, dy. \tag{7.4}$$

as roughness penalty for Ω being a region in $\mathbb{R}^2$ which contains all of the data points (x_i, y_i). They point out that in contrast to the univariate smoothing spline this reduction to Ω makes a difference to the case $\Omega = \mathbb{R}^2$, which was mentioned before.

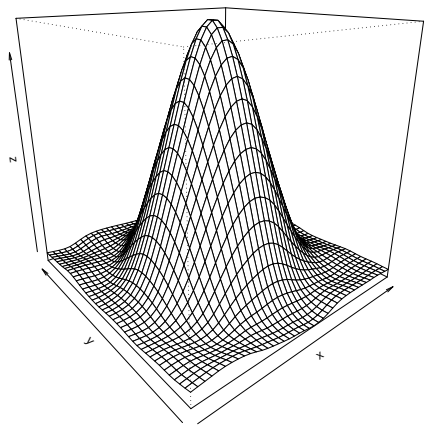


Figure 7.2: Thin plate spline approximation of the noisy cone with automatically determined smoothing parameter ($\lambda = 0.0144$).

7.1.1 Multiresolution as Decision Criterion

For a reasonable approximation of a two dimensional data set by a thin plate spline it is again necessary to specify the smoothing parameter λ which controls the influence of the penalty term. Standard software normally uses the general cross validation function (GCV) to estimate this parameter. The GCV chooses a relatively small value which can cause artefacts in the approximation. Hence we propose to use the multiresolution criteria for the determination. Therefore we combine the approximation of a thin plate smoothing spline with the multiresolution square test which has been described in 6.2.2. Starting with a large parameter λ the computation of the approximation and the multiresolution check are iterated. If the multiresolution criteria are not satisfied on at least one square, λ is reduced for the next thin plate spline approximation. The procedure stops if the multiresolution criteria are satisfied on all squares.
In this context where only a global parameter has to be specified not all of the information of the multiresolution conditions can be used. The information about the regions where the approximation is not good enough and thus does not satisfy the multiresolution constraints is neglected and only the fact is used if or if not all of the conditions are fulfilled. If one of the multiresolution conditions is not satisfied, the global smoothing parameter is reduced in the next step.
Figure 7.2 shows the result of the thin plate smoothing spline using the automatically determined smoothing parameter $\lambda = 0.0144$ for the noisy cone (Figure 7.1). The resulting parameter is three times larger then using GCV. For the computation of the thin plate smoothing spline the `Tps()` function of the library `fields` for the statistic software R was used. Unfortunately it is relatively slow so that we restricted the example to the mentioned data set of dimension 40×40.

7.2 Penalized Triograms

Hansen, Kooperberg and Sardy (1998) introduced the triograms, a class of linear spline models for bivariate smoothing problems. The triogram method estimates functions using piecewise linear, bivariate splines on an adaptively constructed triangulation. Therefore the authors adapted different knot selection strategies. Inspired from this idea Koenker and Mizera (2004) developed a smoothing spline approach to the estimation of triograms. Following the idea of estimating conditional quantile functions[1] (Koenker, Ng and Portnoy, 1994) they were looking for an extension of the total variation based penalty term to two dimensional functions. Requiring orthogonal invariance they show that for such a penalty on a triangulation Δ and g a piecewise linear function on Δ, a constant c can be found, only depending on the choice of the norm such that:

$$J(g, \Omega, \|\cdot\|) = c \sum_k |\nabla g^+_{e_k} - \nabla g^-_{e_k}|_2 \, |e_k|_2. \tag{7.5}$$

with k running over the interior edges of the triangulation, $|e_k|_2$ the Euclidian length of the edge e_k and $|\nabla g^+_{e_k} - \nabla g^-_{e_k}|_2$ the Euclidian length of the difference between gradients of g on the triangles adjacent to e_k. The penalized quantile triograms then solve the following minimization problem over the space of triograms $\mathcal{G}$:

$$\sum_{i=0}^{n} \rho_\tau(z_i - g(x_i, y_i)) + \lambda J(g, \Omega, \|\cdot\|) \tag{7.6}$$

with J the total variation penalty term in (7.5). To compute the solution of such a minimization problem linear programming methods can be used very efficiently.

7.2.1 The Choice of the Smoothing Parameter

This procedure relies on an appropriate choice of the smoothing parameter λ. Koenker and Mizera (2004) refer to different possibilities, e.g., they propose to use the divergence as a measure of the effective dimension of a fit $\hat{g}_\lambda$ for a given λ (cf. Meyer and Woodroofe, 2000) in a model selection criterion, as e.g. the Schwarz criterion, or the Akaike criterion.
The application of the multiresolution square test allows the automatic selection of a smoothing parameter which guarantees an appropriate fit and avoids artefacts. Figure 7.3 shows the result of the penalized triogram fit of the noisy cone using the iteratively chosen smoothing parameter. For the computation again an additional package for the statistic software R, the package `nprq` was used. The function `rqss()` fits a penalized triogram to two dimensional data utilizing special programming techniques for sparse matrices. Being able to compute the triograms the equispaced grid points were varied by adding uniformly distributed random variables.

[1] Here conditional quantile functions are estimated by minimizing $\sum_{i=1}^{n} \rho_\tau(y_i - g(x_i)) + \lambda J(g)$, where $\rho_\tau(u) = u(\tau - I(u < 0))$ and $J(g)$ the total variation of the first derivative of g.

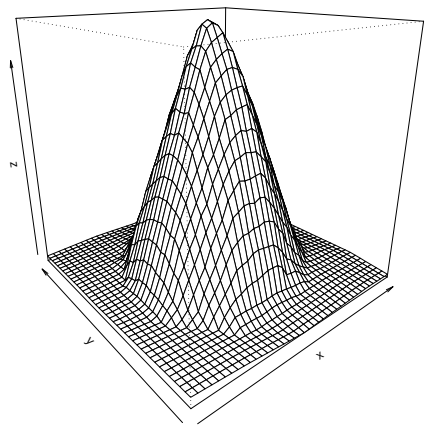


Figure 7.3: Penalized triogram approximation of the noisy cone with automatically determined smoothing parameter ($\lambda = 1.31$).

The comparison between the two bivariate spline extensions, the thin plate smoothing spline and the penalized triogram reveals the difference between the results of these procedures. Whereas the thin plate spline result is very smooth and hence tends to artefacts in the regions where the original function has a sharp edge, the penalized triogram fit is able to approximate the constant part of the function in a better way. Obviously these two methods have very different scopes of application but in both cases using the multiresolution criteria a smoothing parameter is chosen which seems to be very suitable for the particular problem.

7.3 Asymptotics

Considering the situation of twice differentiable functions with bounded partial derivatives we can draw a similar conclusion about the asymptotic behavior for an approximation which satisfies the multiresolution conditions as in the one dimensional situation.

Definition 7.1. *Let $\xi = \xi_{n^2}$ be an equidistant grid on $[0,1]^2$, with grid points (t_i, t_j), $t_i = \frac{i}{n}$, $i, j = 0, \ldots, n$. We say a function $g : [0,1]^2 \to \mathbb{R}$ satisfies the sum condition for ξ_{n^2} and τ, i. e. $g \in S(\xi_{n^2}, \tau)$, if and only if*

$$|\sum_{(t_i,t_j)\in\xi\cap I} g(t_i, t_j)| \leq \sqrt{l_n}\sqrt{2\tau \log(n+1)}.$$

With $l_n = l_n(I) = \#\{(t_i, t_j)|\, 0 \leq i, j \leq n\} \cap I$ for any part $I \subset [0,1]^2$.

Notation: Let g be a function with $g : [0,1]^2 \to \mathbb{R}$ and $g \in C^1([0,1]^2)$, then we write $g^{(1,0)}(x_0, y_0)$ for $\frac{\partial}{\partial x} g(x_0, y_0)$ and analogous $g^{(0,1)}(x_0, y_0)$ for $\frac{\partial}{\partial y} g(x_0, y_0)$.

Definition 7.2. *For $K > 0$ we define*

$$\begin{aligned}\mathcal{W}_2(K) &:= \Big\{ f \mid f : [0,1]^2 \to \mathbb{R}, f \in C^2([0,1]^2), \\ &\qquad \sup_{x\in[0,1]^2} \{ |\frac{\partial^2}{\partial x_\nu x_\mu} f(x)|, \nu, \mu \in \{1,2\}\} \leq K \}.\end{aligned}$$

Theorem 7.1. *For $K_1, K_2 > 0$ let f be a function $f \in \mathcal{W}_2(K_1)$ and $y(x) = f(x) + \varepsilon(x)$, $\varepsilon(x) \sim \mathcal{N}(0,1)$ Let $(\xi_{n^2})_{n\in\mathbb{N}}$ be a sequence of equidistant grids on $[0,1]^2$ and f_n the solution of the approximation procedure, so that $f_n - y \in S(\xi_{n^2}, \tau)$ and $f_n \in \mathcal{W}_2(K_2)$ for each n. Then for each $\alpha > \frac{1}{2}$ and each $\delta > 0$*

$$\lim_{n\to\infty} \mathbb{P}\left(n^{\frac{2}{3}} (\log(n+1))^{-\alpha} \|f - f_n\|_\infty \leq \delta \right) = 1.$$

Proof: For $g : [0,1]^2 \to \mathbb{R}$, $g \in C^2([0,1]^2)$, $x \in [0,1]^2$ and $x + h \in [0,1]^2$ we have

$$g(x_0 + h) = g(x_0) + g^{(1,0)}(x_0)h_1 + g^{(0,1)}(x_0)h_2 + R_1(x_0; h).$$

With $R_1(x_0; h)$ the remainder term at a special point $x_0 + \vartheta h$:

$$R_1(x_0; h) = \left(\left(\frac{1}{2} h_1^2 \frac{\partial^2}{\partial x_1^2} + h_1 h_2 \frac{\partial^2}{\partial x_1 x_2} + \frac{1}{2} h_2^2 \frac{\partial^2}{\partial x_2^2} \right) f \right) (x_0 + \vartheta h)$$

Hence since the second partial derivatives are bounded we have

$$|R_1(x_0, h)| \leq \frac{1}{2} \|h\|_\infty^2 \sum_{\nu,\mu=1}^{2} \sup_{x\in[0,1]^2} \left| \frac{\partial^2}{\partial x_\nu x_\mu} f(x) \right|$$

Set $h_n := f - f_n$, then $h_n \in \mathcal{W}_2(K)$ with $K := K_1 + K_2$. Assuming $f - y \in S(\xi_{n^2}, \tau)$, we know, because of $f_n - y \in S(\xi_{n^2}, \tau)$

$$h_n = f - f_n = f - y - (f_n - y) \in S(\xi_{n^2}, 4\tau)$$

Let be $l, k, \nu, \mu \in \{1, \ldots, n\}$ so that $1 \leq \nu+1+k, \ldots, \nu+l+k \leq n$, $1 \leq \mu+1+k, \ldots, \mu+l+k \leq n$ and $1 \leq \mu+1-k, \ldots, \mu+l-k \leq n$. Then for $i \in \nu+1, \ldots, \nu+l$, $j \in \mu+1, \ldots, \mu+l$ we can write

$$\begin{aligned}&h_n(t_{i+k}, t_{j+k}) = \\ &\quad h_n(t_i, t_j) + h_n^{(1,0)}(t_i, t_j)(t_{i+k} - t_i) + h_n^{(0,1)}(t_i, t_j)(t_{j+k} - t_j) \\ &\quad + R_1((t_i, t_j); (t_{i+k} - t_i, t_{j+k} - t_j)).\end{aligned}$$

Hence we have, because the equidistance of the grid

$$\begin{aligned}&\left(h_n^{(1,0)}(t_i, t_j) + h_n^{(0,1)}(t_i, t_j) \right) \frac{k}{n} = \\ &\quad h_n(t_{i+k}, t_{j+k}) - h_n(t_i, t_j) - R_1((t_i, t_j); (t_{i+k} - t_i, t_{j+k} - t_j)).\end{aligned}$$

Summation over i and j gives

$$\begin{aligned}
&\left|\sum_{i=\nu+1}^{\nu+l}\sum_{j=\mu+1}^{\mu+l}\left(h_n^{(1,0)}(t_i,t_j)+h_n^{(0,1)}(t_i,t_j)\right)\right|\frac{k}{n}\\
&= \left|\sum_{i=\nu+1}^{\nu+l}\sum_{j=\mu+1}^{\mu+l}\left(h_n(t_{i+k},t_{j+k})-h_n(t_i,t_j)\right)-\sum_{i=\nu+1}^{\nu+l}\sum_{j=\mu+1}^{\mu+l}R_1\left((t_i,t_j);\left(\frac{k}{n},\frac{k}{n}\right)\right)\right|\\
&\leq \left|\sum_{i=\nu+1}^{\nu+l}\sum_{j=\mu+1}^{\mu+l}h_n(t_{i+k},t_{j+k})\right|+\left|\sum_{i=\nu+1}^{\nu+l}\sum_{j=\mu+1}^{\mu+l}h_n^{(0,1)}(t_i,t_j)\right|\\
&\quad+\left|\sum_{i=\nu+1}^{\nu+l}\sum_{j=\mu+1}^{\mu+l}R_1\left((t_i,t_j);\left(\frac{k}{n},\frac{k}{n}\right)\right)\right|\\
&\leq 2l\sqrt{8\tau\log(n+1)}+l^2 2K\left(\frac{k}{n}\right)^2.
\end{aligned}$$

In the same way we get

$$\begin{aligned}
&h_n(t_{i+k},t_{j-k}) =\\
&\quad h_n(t_i,t_j)+h_n^{(1,0)}(t_i,t_j)(t_{i+k}-t_i)+h_n^{(0,1)}(t_i,t_j)(t_{j-k}-t_j)\\
&\quad +R_1((t_i,t_j);(t_{i+k}-t_i,t_{j-k}-t_j))
\end{aligned}$$

and hence

$$\begin{aligned}
&\left(h_n^{(1,0)}(t_i,t_j)-h_n^{(0,1)}(t_i,t_j)\right)\frac{k}{n} =\\
&\quad h_n(t_{i+k},t_{j-k})-h_n(t_i,t_j)-R_1((t_i,t_j);(t_{i+k}-t_i,t_{j-k}-t_j)).
\end{aligned}$$

Thus we have

$$\begin{aligned}
&\left|\sum_{i=\nu+1}^{\nu+l}\sum_{j=\mu+1}^{\mu+l}\left(h_n^{(1,0)}(t_i,t_j)-h_n^{(0,1)}(t_i,t_j)\right)\right|\frac{k}{n}\\
&\leq \left|\sum_{i=\nu+1}^{\nu+l}\sum_{j=\mu+1}^{\mu+l}h_n(t_{i+k},t_{j-k})\right|+\left|\sum_{i=\nu+1}^{\nu+l}\sum_{j=\mu+1}^{\mu+l}h_n^{(0,1)}(t_i,t_j)\right|\\
&\quad+\left|\sum_{i=\nu+1}^{\nu+l}\sum_{j=\mu+1}^{\mu+l}R_1\left((t_i,t_j);\left(\frac{k}{n},-\frac{k}{n}\right)\right)\right|\\
&\leq 2l\sqrt{8\tau\log(n+1)}+2K\left(\frac{k}{n}\right)^2.
\end{aligned}$$

Using the unequality $|2a| = |a+b+a-b| \leq |a+b|+|a-b|$ respectively $|2b| = |a+b-(a-b)| \leq |a+b|+|a-b|$ for the sums of the first derivatives we obtain

$$\left| \sum_{i=\nu+1}^{\nu+l} \sum_{j=\mu+1}^{\mu+l} h_n^{(1,0)}(t_i, t_j) \right| \frac{k}{n} \leq 2l\sqrt{8\tau \log(n+1)} + 2K \left(\frac{k}{n}\right)^2$$

and

$$\left| \sum_{i=\nu+1}^{\nu+l} \sum_{j=\mu+1}^{\mu+l} h_n^{(0,1)}(t_i, t_j) \right| \frac{k}{n} \leq 2l\sqrt{8\tau \log(n+1)} + 2K \left(\frac{k}{n}\right)^2.$$

Now we choose n large and $\mu_n := n^{-\tilde{q}}$ so that for every $x \in (0,1)^2$ we can assume that $[x_1 - 3\mu_n, x_1 + 3\mu_n] \times [x_2 - 3\mu_n, x_2 + 3\mu_n] \subset [0,1]^2$. Then we define $I_n(x) := [x_1 - \mu_n, x_1 + \mu_n] \times [x_2 - \mu_n, x_2 + \mu_n]$, and ν_n, ρ_n and $l_n \in \mathbb{N}$ so that $I_n(x) \cap \xi_{n^2} = \{\frac{\nu_n+1}{n}, \ldots, \frac{\nu_n+l_n}{n}\} \times \{\frac{\rho_n+1}{n}, \ldots, \frac{\rho_n+l_n}{n}\}$ and hence $\#\{I_n(x) \cap \xi_{n^2}\} = l_n^2$. Now we can choose $k_n := l_n$ and have $1 \leq \nu_n+1+k_n, \ldots \nu_n+l_n+k_n \leq n$, $1 \leq \rho_n+1+k_n, \ldots \rho_n+l_n+k_n \leq n$ and $1 \leq \rho_n+1-k_n, \ldots \rho_n+l_n-k_n \leq n$. For each x now then it is

$$\begin{aligned} h_n(x) &= h_n(t_i, t_j) + h_n^{(1,0)}(t_i, t_j)(x_1 - t_i) + h_n^{(0,1)}(t_i, t_j)(x_2 - t_j) \\ &\quad + R_1((t_i, t_j); (x_1 - t_i, x_2 - t_j)). \end{aligned}$$

Summing over i and j and using $|I_n|$ as the edge length of I_n we obtain

$$\begin{aligned} l_n^2 |h_n(x)| &\leq \left| \sum_{i=i_n+1}^{i_n+l_n} \sum_{j=j_n+1}^{j_n+l_n} h_n(t_i, t_j) \right| + \left| \sum_{i=i_n+1}^{i_n+l_n} \sum_{j=j_n+1}^{j_n+l_n} h_n^{(1,0)}(t_i, t_j) \right| |I_n| \\ &\quad + \left| \sum_{i=i_n+1}^{i_n+l_n} \sum_{j=j_n+1}^{j_n+l_n} h_n(t_i, t_j) \right| |I_n| + l_n^2 2K |I_n|^2 \\ &\leq l_n\sqrt{8\tau \log(n+1)} + \frac{n}{k_n}|I_n| \left(4l_n\sqrt{8\tau \log(n+1)} + l_n^2 4K \left(\frac{l_n}{n}\right)^2 \right) \\ &\quad + l_n^2 2K |I_n|^2 \\ &\leq l_n\sqrt{8\tau \log(n+1)} + \frac{n}{l_n} 2\mu_n \left(4l_n\sqrt{8\tau \log(n+1)} + l_n^2 4K \left(\frac{l_n}{n}\right)^2 \right) \\ &\quad + l_n^2 2K (2\mu_n)^2 \\ &\leq l_n\sqrt{8\tau \log(n+1)} + 4\sqrt{8\tau \log(n+1)} n 2\mu_n + 4K \frac{l_n^3 2\mu_n}{n} \\ &\quad + l_n^2 2K (2\mu_n)^2. \end{aligned}$$

Dividing by l_n^2 and using $\mu_n n \leq l_n \leq 2\mu_n n$ we get

$$\begin{aligned}
|h_n(x)| &\leq \frac{1}{l_n}\sqrt{8\tau\log(n+1)} + 4\sqrt{8\tau\log(n+1)}\frac{2\mu_n n}{l_n^2} + 8K\frac{\mu_n l_n}{n} + 2K(2\mu_n)^2 \\
&\leq \frac{1}{\mu_n n}\sqrt{8\tau\log(n+1)} + 8\sqrt{8\tau\log(n+1)}\frac{1}{\mu_n n} + 16K\mu_n^2 + 8K\mu_n^2 \\
&\leq \frac{9}{\mu_n n}\sqrt{8\tau\log(n+1)} + 24K\mu_n^2.
\end{aligned}$$

With $\mu_n = n^{-\frac{1}{3}}$ this implies

$$|h_n(x)| \leq 9n^{-\frac{2}{3}}\sqrt{8\tau\log(n+1)} + 24Kn^{-\frac{2}{3}}.$$

For $\alpha > \frac{1}{2}$ we then have

$$\begin{aligned}
n^{\frac{2}{3}}(\log(n+1))^{-\alpha}|h_n(x)| &\leq \sqrt{8\tau}\log(n+1)^{-(\alpha-\frac{1}{2})} + 24Kn^{-\frac{2}{3}}(\log(n+1))^{-\alpha} \\
&\longrightarrow 0 \quad (n \longrightarrow \infty).
\end{aligned}$$

That means for each $\delta > 0$ we find $N_\delta \in \mathbb{N}$ so that for each $n \geq N_\delta$:

$$n^{\frac{2}{3}}(\log(n+1))^{-\alpha}\|h_n\|_\infty \leq \delta.$$

This gives

$$\mathbb{P}\left(\{f - y \in S(\xi_{n^2}, \tau)\}\right) \leq \mathbb{P}\left(n^{\frac{2}{3}}(\log(n+1))^{-\alpha}\|f - f_n\|_\infty \leq \delta\right)$$

and hence

$$\mathbb{P}\left(n^{\frac{2}{3}}(\log(n+1))^{-\alpha}\|f - f_n\|_\infty \leq \delta\right) \longrightarrow 1 \quad (n \longrightarrow \infty).$$

□

7.4 Remark

The previous chapters showed that the multiresolution analysis of the residuals combined with nonparametric regression methods give good results, not only in one but also in two dimensional situations. This automatic determination of locally defined smoothing parameters improves the local flexibility of the approximations. But also the global parameter selection can be very useful in the context of thin plate splines and penalized triograms, as the examples show.
However, there are still open questions. For example it would be interesting to know how the approximation can be improved for data sets which obviously contain discontinuouities. Also the problem of smoothing under monotonicity constraints is not finally solved. In this context it would be interesting to develop a procedure which provides nice properties like differentiability for the smooth parts but at the same time allows rapid changes.

Obviously, also the bivariate procedures raise questions. In order to increase the size of the data sets it would be helpful to develop programs, which reduce the computing time and the memory demand. Furthermore it would be interesting to consider also bivariate data which are not situated on a grid. Therefore the residual analysis needs to be changed. A localization of the smoothing parameters would improve the flexibility of two dimensional splines. A satisfactory determination of such local parameters would provide the possibility to approximate smooth surfaces which are sensitive enough to adapt also small features. At the same time artefacts would be avoided.

Appendix A

Source Codes

A.1 Kernel Estimator

A.1.1 R Code

```
KERNMR<-function(x,y, tau=2.5, kern=3,alpha=0.8, mr=1,loc=1,sigma,bandw,nit)
{
  n <- length(y)
  reg <- double(n)
  res <- double(n)
  if(missing(sigma)){
    b <- double(n-1)
    for (i in 1:(n-1)){
      b[i]<-(y[i]-y[i+1])/sqrt(2)
    }
    sigma <- mad(b,center=0)
  }
  thresh <- sigma*sqrt(tau*log(n))
  if(missing(bandw))
    bandw <- rep(abs(max(x)-min(x)),le=n)
  else
    if(length(bandw)!=n){
      stop("Bandwidth and data not compatible!")
    }
  if((mr==2)&&(missing(nit))){
    print("Number of iterations not specified!")
    break
  }
  else
    if(missing(nit))
      nit=1
  zzz<-.C("KERNMR",
          as.double(y),
          as.integer(n),
          as.double(thresh),
          as.double(x),
          as.double(bandw),
          as.double(reg),
          as.double(res),
          as.double(alpha),
          as.integer(kern),
          as.integer(mr),
          as.integer(nit),
          as.integer(loc))
 list(bandw=zzz[[5]], reg=zzz[[6]], sigma=sigma, thresh=thresh)
}


multires <- function(y,thresh, firstwidth=1)
  {
    n <- length(y)
    zzz <- .C("multiwdwr", as.double(y),
                           as.integer(n),
                           as.double(thresh),
                           as.integer(firstwidth))
    zzz[[1]]
  }
```

A.1.2 C Code

```
#include "math.h"
#include "stdlib.h"
#include "stdio.h"

/* Uniform */

double kern1(double u)
{
  double w;
  if ((u >= -1) && (u <= 1))
    w = 0.5;
  else w = 0;
  return w;
}


/* Triangle */

double kern2(double u)
{
  double w;
  if((u >= -1) && (u < 0))
    w = 1 - (-u);
  else{
    if ((u >= 0) && (u < 1))
      w = 1 -u;
    else w = 0;
  }
  return w;
}

/* Epanechnikov */

double kern3 (double u)
{
  double w;
  if ((u >= -1) && (u <= 1))
    w = 0.75*(1-u*u);
  else w = 0;
  return w;
}


/* Quartic */

double kern4 (double u)
{
  double w;
  if ((u >= -1) && (u <= 1))
    w = 0.9375*(1-u*u)*(1-u*u);
  else w = 0;
  return w;
}

/* Triweight */

double kern5 (double u)
{
  double w;
  if ((u >= -1) && (u <= 1))
    w = 1.09375*(1-u*u)*(1-u*u)*(1-u*u);
  else w = 0;
  return w;
}


/* Cosinus */

double kern6 (double u)
{
  double w;
  if ((u >= -1) && (u <= 1))
    w = 0.7853982 * cos(1.5707963*u);
  else w = 0;
  return w;
}



void multiwdwr(double *y,
               int *n,
               double *thresh,
               int *firstwidth)
{
int j, actwidth, leftind, rightind;
```

```
double *ysum;

ysum=malloc((*n+1)*sizeof(double));
ysum[0]=0;
for(j=1;j<=*n;j++)
  ysum[j]=ysum[j-1]+y[j-1];
for(j=0;j<*n;j++)
  y[j]=0.0;

for(actwidth=*firstwidth;actwidth<=*n;actwidth*=2)
  for(leftind=0,rightind=actwidth;leftind<*n;
            leftind=rightind,rightind+=actwidth)
    {
    if(rightind>*n) rightind= *n;
    if(fabs((ysum[rightind]-ysum[leftind])/
         sqrt((double)(rightind-leftind)))>*thresh)
      for(j=leftind;j<rightind;j++)
        y[j]=1.0;
    }

free(ysum);
}


void NWKE(double *y,
          double *x,
          double *h,
          double *erg,
          double *otto2,
          int *n,
          double (*kernfkt)())
{
  int i,j,nh[*n];
  double Kh;
  double KKK;
  int links, rechts;
  for(j=0;j<*n;j++){
    if(otto2[j]==1.0){
      KKK=0.0;
      erg[j]=0.0;
      nh[j]=*n*h[j];
      if(0>j-nh[j])
        links=0;
      else links=j-nh[j];
      if(*n<j+nh[j])
        rechts=*n-1;
      else rechts=j+nh[j];
      for (i=links;i<=rechts;i++){
        Kh=kernfkt((x[j]-x[i])/h[j])*1/h[j];
        KKK+=Kh;
        erg[j]+=Kh*y[i];
      }
      if(KKK==0.0)
        erg[j]=0.0;
      else
        erg[j]=erg[j]/KKK;
    }
  }
}


void NWKEconst(double *y,
               double *x,
               double h,
               double *erg,
               int *n,
               double (*kernfkt)())
{
  int i,j,nh;
  double Kh;
  double KKK;
  int links, rechts;
  for(j=0;j<*n;j++){
      KKK=0.0;
      erg[j]=0.0;
      nh=*n*h;
      if(0>j-nh)
        links=0;
      else links=j-nh;
      if(*n<j+nh)
        rechts=*n-1;
      else rechts=j+nh;
      for (i=links;i<=rechts;i++){
          Kh=kernfkt((x[j]-x[i])/h)*1/h;
        KKK+=Kh;
        erg[j]+=Kh*y[i];
      }
      if(KKK==0.0)
```

```
        erg[j]=0.0;
      else
        erg[j]=erg[j]/KKK;
  }
}



void KERNMR(double *y,
            int *n,
            double *thresh,
            double *x,
            double *bandw,
            double *reg,
            double *res,
            double *alpha,
            int *kern,
            int *mr,
            int *cb,
            int *loc)
{
  int i,j,bk, otto1, *firstwidth, counter;
  double *ifreg,bw;
  double (*kernfkt)();

  switch (*kern)
    {
    case 1 : kernfkt = kern1; break;
    case 2 : kernfkt = kern2; break;
    case 3 : kernfkt = kern3; break;
    case 4 : kernfkt = kern4; break;
    case 5 : kernfkt = kern5; break;
    case 6 : kernfkt = kern6; break;
    default: printf("Fehler in der Wahl der Kernfunktion\n");
    return;
    }
  otto1=0;
  counter=0;
  ifreg=malloc(sizeof(double)**n);
  firstwidth=malloc(sizeof(double));
  *firstwidth=1;
  for (i=0;i<*n;i++){
    ifreg[i]=1;
   }
  if(*loc!=1)
    bw=bandw[0];
  if(*mr==1)
    do {
      otto1=0;
      counter+=1;
      printf("counter: %i\n", counter);
      if(*loc==1)
        NWKE(y, x, bandw, reg, ifreg, n, kernfkt);
      else
        NWKEconst(y, x, bw, reg, n, kernfkt);
      for(i=0;i<*n;i++){
        res[i]=reg[i]-y[i];
        ifreg[i]=res[i];
      }
      multiwdwr(ifreg, n,thresh,firstwidth);
      if(*loc==1){
        for(i=0;i<*n;i++){
          if(fabs(ifreg[i]-1.0)<1e-16){
            otto1=1;
            bandw[i]=*alpha*bandw[i];
          }
        }
      }
      else{
        i=0;
        bk=0;
        do{
          if(fabs(ifreg[i]-1.0)<1e-16){
            otto1=1;
            bk=1;
            bw=*alpha*bw;
          }
          if(i<(*n-1))
            i+=1;
          else
            bk=1;
        }while(bk==0);
      }
    }while(otto1==1);
  else
    if(*mr==2)
      do {
        otto1=0;
```

```
        counter+=1;
        printf("counter: %i\n", counter);
        NWKE(y, x, bandw, reg, ifreg, n, kernfkt);
        for(i=0;i<*n;i++){
          res[i]=reg[i]-y[i];
          ifreg[i]=res[i];
        }
        multiwdwr(ifreg, n,thresh,firstwidth);
        if(counter<*cb)
          if(*loc==1){
            for(i=0;i<*n;i++){
              if(fabs(ifreg[i]-1.0)<1e-16){
                otto1=1;
                bandw[i]=*alpha*bandw[i];
              }
            }
          }
          else{
            i=0;
            bk=0;
            do{
              if(fabs(ifreg[i]-1.0)<1e-16){
                otto1=1;
                bk=1;
                bw=*alpha*bw;
              }
              if(i<(*n-1))
                i+=1;
              else
                bk=1;
            }while(bk==0);
          }
      }while(counter<*cb);
    else
      NWKE(y, x, bandw, reg, ifreg, n, kernfkt);
  if(*loc!=1)
    for(i=0;i<*n;i++)
      bandw[i]=bw;
  free(ifreg);
  free(firstwidth);
}
```

A.2 Local Polynomials

A.2.1 R Code

```
lokpolmr<-function(x,y,p, tau=2.5, alpha=0.8, mr=1, bandsmooth=0, bandw, nit)
  {
    n<-length(y)
    b <- double(n-1)
    for (i in 1:(n-1)){
      b[i]<-(y[i]-y[i+1])/sqrt(2)
    }
    sigma <- mad(b,center=0)
    print(sigma)
    thresh <- sigma*sqrt(tau*log(n))
    if(missing(bandw))
      bandw <- rep(abs(max(x)-min(x)),le=n)
    else
      if(length(bandw)!=n)
        stop("Bandwidth and data not compatible!")
    if((mr==2)&&(missing(nit)))
      stop("Number of iterations not specified!")
    else
      if(missing(nit))
        nit=1
    if((bandsmooth!=0)&&(mr!=1))
      stop("Bandwidth smoothing only possible in combination with MR!")
    XXX<-.C("lokpolmr",
            as.double(x),
            as.double(y),
            as.integer(n),
            as.integer(p),
            as.double(bandw),
            reg=double(n),
            bandw1=double(n),
            reg1=double(n),
            as.double(thresh),
            as.double(sigma),
            as.double(alpha),
            as.integer(mr),
```

```
            as.integer(nit),
            as.integer(bandsmooth))
    list(bandw=XXX[[5]],
         bandw1=XXX$bandw1,
         reg=XXX$reg,
         reg1=XXX$reg1,
         sigma=sigma,
         thresh=thresh)
  }
```

A.2.2 C Code

```
#include "math.h"
#include "stdlib.h"
#include "stdio.h"

void multiwdwr(double *y,
               int *n,
               double *thresh,
               int *firstwidth)
{
int j, actwidth, leftind, rightind;
double *ysum;

 ysum=malloc((*n+1)*sizeof(double));
 ysum[0]=0;
 for(j=1;j<=*n;j++)
   ysum[j]=ysum[j-1]+y[j-1];
 for(j=0;j<*n;j++)
   y[j]=0.0;

 for(actwidth=*firstwidth;actwidth<=*n;actwidth*=2)
   for(leftind=0,rightind=actwidth;leftind<*n;
                 leftind=rightind,rightind+=actwidth)
     {
       if(rightind>*n) rightind= *n;
       if(fabs((ysum[rightind]-ysum[leftind])/
                 sqrt((double)(rightind-leftind)))> *thresh)
         for(j=leftind;j<rightind;j++)
           y[j]=1.0;
     }

 free(ysum);
}



void QRZerl(double *A,
            int *m,
            int *n,
            double *b,
            double *Y,
            double *B)
{
  int i,j,k,*e1;
  double normx,normvk,*x,*vk,*PROD1,*PROD2, Const;
  for(k=0;k<*n;k++){
    x=malloc((*m-k)*sizeof(double));
    e1=malloc((*m-k)*sizeof(int));
    vk=malloc((*m-k)*sizeof(double));
    PROD1=malloc((*m-k)*sizeof(double));
    PROD2=malloc((*n-k)*(*m-k)*sizeof(double));
    e1[0]=1;
    for(i=1;i<(*m-k);i++){
      e1[i]=0;
    }
    normx=0.0;
    for(i=k;i<*m;i++){
      x[(i-k)]=A[i**n+k];
      normx=normx+x[(i-k)]*x[(i-k)];
    }
    normx=sqrt(normx);
    normvk=0.0;
    if (x[1]==0)
      for(i=0;i<(*m-k);i++){
        vk[i]=-normx*e1[i]-x[i];
        normvk=normvk+vk[i]*vk[i];
      }
    else
      if(x[0]<0)
        for(i=0;i<(*m-k);i++){
          vk[i]=-normx*e1[i]+x[i];
          normvk=normvk+vk[i]*vk[i];
        }
      else
```

```
        for(i=0;i<(*m-k);i++){
          vk[i]=normx*e1[i]+x[i];
          normvk=normvk+vk[i]*vk[i];
        }
    normvk=sqrt(normvk);
    for(i=0;i<(*m-k);i++)
      vk[i]=vk[i]/normvk;
    for(i=0;i<(*n-k);i++){
      PROD1[i]=0;
      for(j=0;j<(*m-k);j++)
        PROD1[i]+=vk[j]*A[(j+k)**n+(i+k)];
    }
    for(i=0;i<(*m-k);i++)
      for(j=0;j<(*n-k);j++)
        PROD2[i*(*n-k)+j]=2*vk[i]*PROD1[j];
    for(i=0;i<(*m-k);i++)
      for(j=0;j<(*n-k);j++)
        A[(i+k)**n+(j+k)]=A[(i+k)**n+(j+k)]-PROD2[i*(*n-k)+j];
    Const=0.0;
    for(i=0;i<(*m-k);i++)
      Const+=vk[i]*b[(i+k)];
    for(i=0;i<(*m-k);i++)
      b[i+k]=b[i+k]-2*vk[i]*Const;
    free(x);
    free(e1);
    free(vk);
    free(PROD1);
    free(PROD2);
  }
  for(i=0;i<*n;i++){
    B[i]=b[i];
    for(j=0;j<*n;j++){
      Y[i**n+j]=A[i**n+j];
    }
  }
}

void linAusgl (double *R,
               double *QTb,
               int *n,
               double *x)
{
  int i,j,k;
  double h;
  for(j=(*n-1);j>=0;j=j-1){
    R[j**n+j]=1.0/R[j**n+j];
    for(i=(*n-1);i>=(j+1);i=i-1){
      h=0.0;
      for(k=(j+1);k<=i;k++){
        h=h-R[j**n+k]*R[k**n+i];
      }
      R[j**n+i]=h*R[j**n+j];
    }
  }
  for(i=0;i<*n;i++){
    x[i]=0.0;
    for(j=0;j<*n;j++)
      x[i]+=R[i**n+j]*QTb[j];
  }
}


void MatBel(double *x,
            int *m,
            int *p,
            double *A)
{
  int i,j,k;
  for(j=0;j<*m;j++)
    for(i=0;i<*p;i++){
      A[j**p+i]=1.0;
      for(k=1;k<=i;k++)
        A[j**p+i]=A[j**p+i]*x[j];
    }
}


void lokReg (double *A,
             int j,
             int *n,
             double *x,
             double *yy)
{
  int i;
  *yy=0.0;
  for(i=0;i<*n;i++){
    *yy+=A[i]*x[i];
  }
```

```
}


/* Epanechnikov */

double kern (double u)
{
  double w;
  if ((u >= -1) && (u <= 1)){
    w = 0.75*(1-u*u);
  }
  else{
    w = 0;
  }
  return w;
}


void Gewichte (double *x,
               double x0,
               int *m,
               double bandw,
               double *W)
{
  int i;
  double kern();
  for(i=0;i<*m;i++){
    W[i]=(1.0/bandw)*kern((x[i]-x0)/(bandw));
  }
}


void Wichtung (double *W,
               double *A,
               double *YYY,
               int *m,
               int *n,
               double *a,
               double *y)
{
  int i,j;
  for(i=0;i<*m;i++){
      if(W[i]!=0.0){
     W[i]=sqrt(W[i]);
    }
    for(j=0;j<*n;j++){
      a[i**n+j]=A[i**n+j]*W[i];
    }
    y[i]=YYY[i]*W[i];
  }
}

void lkp(double *x,
         double *YYY,
         int *m,
         int *n,
         double *bandw,
         double *reg,
         double *AAA,
         double *vergl,
         int *ZZZ)
{
  int i,j,J,k;
  double *aaa,*www,*rrr,*qtb,*xx,*hhh,*aaaJ, *y, lll;
  if(*ZZZ!=1){
    for(J=0;J<*m;J++){
      if(vergl[J]==1.0){
        aaa=malloc(*m**n*sizeof(double));
        y=malloc(*m*sizeof(double));
        www=malloc(*m*sizeof(double));
        aaaJ=malloc(*n*sizeof(double));
        rrr=malloc(*n**n*sizeof(double));
        qtb=malloc(*n*sizeof(double));
        xx=malloc(*n*sizeof(double));
        hhh=malloc(1*sizeof(double));
        Gewichte(x,x[J],m,bandw[J],www);
        Wichtung(www,AAA,YYY,m,n,aaa,y);
        for(i=0;i<*n;i++){
          aaaJ[i]=AAA[J**n+i];
        }
        QRZerl(aaa,m,n,y,rrr,qtb);
        linAusgl(rrr,qtb,n,xx);
        lokReg(aaaJ,J,n,xx,hhh);
        reg[J]=*hhh;
        free(aaa);
        free(www);
        free(aaaJ);
        free(rrr);
```

```
        free(qtb);
        free(xx);
        free(hhh);
        free(y);
      }
    }
  }
  else{
    J=(*m-1)/2;
    aaa=malloc(*m**n*sizeof(double));
     y=malloc(*m*sizeof(double));
     rrr=malloc(*n**n*sizeof(double));
     qtb=malloc(*n*sizeof(double));
     xx=malloc(*n*sizeof(double));
     hhh=malloc(1*sizeof(double));
     aaaJ=malloc(*m**n*sizeof(double));
    for(j=0;j<*m;j++){
       y[j]=YYY[j];
      for(i=0;i<*n;i++){
        aaaJ[j**n+i]=AAA[j**n+i];
        aaa[j**n+i]=AAA[j**n+i];
      }
     }

    QRZerl(aaa,m,n,y,rrr,qtb);
    linAusgl(rrr,qtb,n,xx);
    for(j=0;j<*m;j++){
      reg[j]=0.0;
      for(i=0;i<*n;i++){
        lll=1.0;
        for(k=0;k<*n;k++){
          lll=lll*aaaJ[j**n+i];
        }
        reg[j]+=lll*xx[i];
      }
    }

    free(aaa);
     free(rrr);
     free(qtb);
     free(hhh);
     free(aaaJ);
     free(xx);
     free(y);

  }
}



void lokpolanp(double *x,
               double *y,
               int *m,
               int *n,
               double *reg,
               double *bandw,
               int *STP,
               double *grenze,
               int *ZZZ,
               int *mr,
               int *nit,
               double *alpha)
{
  int i, counter, STOPP, *firstwidth;
  double *AAA, *res;
  *STP=0;
  firstwidth=malloc(1*sizeof(int));
  AAA=malloc(*m**n*sizeof(double));
  res=malloc(*m*sizeof(double));
  *firstwidth=1;
  for(i=0;i<*m;i++) res[i]=1.0;
  MatBel(x,m,n,AAA);
  counter=0;
  if(*mr==0){
    *ZZZ=0;
    lkp(x,y,m,n,bandw,reg,AAA,res, ZZZ);
  }
  else
    if(*mr==2)
      do{
        STOPP=0;
        *ZZZ=0;
        counter+=1;
        printf("counter: %i\n", counter);
        lkp(x,y,m,n,bandw,reg,AAA,res, ZZZ);
        for(i=0;i<*m;i++){
```

```
          res[i]=y[i]-reg[i];
        }
        multiwdwr(res, m, grenze,firstwidth);
        if(counter<*nit)
          for(i=0;i<*m;i++){
            if(res[i]==1.0){
              bandw[i]=*alpha*bandw[i];
              if(counter==1){
                *STP=1;
                *ZZZ=0;
              }
              STOPP=1;
              if((bandw[i]**m)<*n-1){
                STOPP=0;
                counter=*nit;
              }
            }
          }
        if(STOPP==0)
          counter=*nit;
      }while(counter<*nit);

    else
      do{
        *ZZZ=0;
        STOPP=0;
        counter+=1;
        printf("counter: %i\n", counter);
        lkp(x,y,m,n,bandw,reg,AAA,res, ZZZ);
        for(i=0;i<*m;i++){
          res[i]=y[i]-reg[i];
        }
        multiwdwr(res, m, grenze,firstwidth);

        for(i=0;i<*m;i++){
          if(res[i]==1.0){
            bandw[i]=*alpha*bandw[i];
            if(counter==1){
              *STP=1;
              *ZZZ=0;
            }
            STOPP=1;
            if((bandw[i]**m)<*n-1){
              STOPP=0;
            }
          }
        }
      }while(STOPP==1);
  free(AAA);
  free(firstwidth);
  free(res);
}



void Schaetzer(double *a,
               double *x,
               int *laengea,
               double *h,
               double *mh)
{
  int i, j;
  double fh[*laengea], rh[*laengea], y[*laengea], ky[*laengea];

  for (j = 0; j < *laengea; j++){
     fh[j]=0;
     rh[j]=0;
     mh[j]=0;
    for (i = 0; i < *laengea; i++){
      y[i]=0;
      y[i]=(x[i]-x[j])/h[j];
      y[i]= kern(y[i]);
      fh[j] += y[i];
      rh[j] += y[i] * a[i];
    }
   if (fh[j]==0)
     mh[j]=0;
    else
     mh[j] = (rh[j])/(fh[j]);
  }
}


void lokpolmr(double *x,
              double *y,
              int *m,
              int *p,
              double *bandw ,
```

```
              double *reg,
              double *bandw1,
              double *reg1,
              double *grenze,
              double *sn,
              double *alpha,
              int * mr,
              int * nit,
              int *bandsmooth)
{
  double M, bbb;
  double *bwbandw, *bwerg, TTT;
  int *STP, i,*n, SSS, *ZZZ;
  bwbandw=malloc(*m*sizeof(double));
   bwerg=malloc(*m*sizeof(double));
   STP=malloc(1*sizeof(int));
   ZZZ=malloc(1*sizeof(int));
   n=malloc(1*sizeof(int));
   SSS=0;
   *ZZZ=0;
   TTT=0.0;
   *n=*p+1;
   M=*m;
   *ZZZ=1;
  if(*bandsmooth==0)
    lokpolanp(x,y,m,n,reg,bandw,STP,grenze, ZZZ, mr, nit, alpha);
  else{
    lokpolanp(x,y,m,n,reg,bandw,STP,grenze, ZZZ, mr, nit, alpha);
    for(i=0;i<*m;i++){
      reg1[i]=reg[i];
      bandw1[i]=bandw[i];
      bwbandw[i]=0.04;
    }
    if(*STP==1){
      do{
        for(i=0;i<*m;i++){
          bwbandw[i]=bwbandw[i]*0.5;
        }
        Schaetzer(bandw,x,m,bwbandw, bwerg);
        lokpolanp(x,y,m,n,reg,bwerg,STP,grenze, ZZZ, mr, nit, alpha);
        if(*STP==0){
          for(i=0;i<*m;i++){
            bandw[i]=bwerg[i];
          }
          SSS=1;
        }
         else{
           bbb=0.0;
           for(i=0;i<*m;i++){
             bbb+=(bandw[i]-bwerg[i]);
             bandw[i]=bwerg[i];
           }
           if(bbb==0){
             for(i=0;i<*m;i++){
               bandw[i]=bwerg[i];
             }
             SSS=1;
           }
         }
      }while(SSS!=1);
    }
  }

  free(bwbandw);
  free(bwerg);
  free(STP);
  free(n);
  free(ZZZ);

}
```

A.3 Weighted Smoothing Splines

A.3.1 R Code

```
SPLlokanp <- function(x,y,tau=2.5, weights,sigma, thresh, mr=1,glob=0,nit)
  {
    n <- length(y)
    if(length(x)!=n)
      stop("x and y length differ")
    b <- double((n-1))
```

```
    for (i in 1:(n-1)){
      b[i]<- (y[i]-y[i+1])/sqrt(2)
    }
    if(missing(sigma))
      sigma <- mad(b,center=0)
    print(c("sigma:", sigma))
    if(missing(thresh))
      thresh <- sigma*sqrt(tau*log(n))
    print(c("thresh:", thresh))
    if(missing(weights)){
      w <- 1/n*1/(max(x)-min(x))^2
      weights <- rep(w,le=n)
    }
    else{
      if(length(weights)!=n)
        stop("y and weights lengths differ")
    }
    if((mr==2)&&(missing(nit))){
      print("Number of iterations not specified!")
      break
    }
    else
      if(missing(nit))
        nit=1
    XXX <- .C("SPLlokanp",
              as.double(x),
              as.double(y),
              as.integer(n),
              as.double(weights),
              as.double(thresh),
              reg=double(n),
              as.integer(glob),
              as.integer(mr),
              as.integer(nit))
    list(weights=XXX[[4]], reg=XXX$reg, thresh=thresh, sigma=sigma)
  }
```

A.3.2 C Code

```
#include <math.h>
#include <stdio.h>


void multiwdwr(double *y,
               int *n,
               double *thresh,
               int *firstwidth)
{
  int j, actwidth, leftind, rightind;
  double *ysum;

  ysum=(double *)malloc((*n+1)*sizeof(double));
  ysum[0]=0;
  for(j=1;j<=*n;j++)
    ysum[j]=ysum[j-1]+y[j-1];
  for(j=0;j<*n;j++)
    y[j]=0.0;
  for(actwidth=*firstwidth;actwidth<=*n;actwidth*=2)
    for(leftind=0,rightind=actwidth;leftind<*n;
                leftind=rightind,rightind+=actwidth){
      if(rightind>*n) rightind= *n;
      if(fabs((ysum[rightind]-ysum[leftind])/
                sqrt((double)(rightind-leftind)))>*thresh)
        for(j=leftind;j<rightind;j++)
          y[j]=1.0;
    }
  free(ysum);
}


void cholesky(double *A,
              int *n,
              double *L,
              int *FFF)
{
  double a[*n][3],l[*n][3];
  int i,j,k,p;
  a[0][0]=0.0;
  a[1][0]=0.0;
  a[0][1]=0.0;
  l[0][0]=0.0;
  l[1][0]=0.0;
  l[0][1]=0.0;
  for(k=0;k<*n;k++){
```

```
    a[k][2]=A[k*(*n)+k];
    if(k>0){
      a[k][1]=A[k**n+(k-1)];
      if(k>1)
        a[k][0]=A[k**n+(k-2)];
    }
  }
  for(k=0;k<*n;k++){
    if(a[k][2]<=0.0){
      printf("a[%i][2]: %lf\n", k,a[k][2]);
      printf("Nicht lösbar!\n");
      *FFF=1;
      break;
    }
    l[k][2]=sqrt(a[k][2]);
    if((k+2)<=*n) p=k+2;
    else p=*n;
    for(i=k+1;i<=p;i++){
      l[i][k-i+2]=a[i][k-i+2]/l[k][2];
      for(j=k+1;j<=i;j++)
        a[i][j-i+2]=a[i][j-i+2]-l[i][k-i+2]*l[j][k-j+2];
    }
  }
  for(k=0;k<*n;k++){
    L[k*(*n)+k]=l[k][2];
    if(k<(*n-1)){
      L[(k+1)**n+k]=l[k+1][1];
      if(k<(*n-2))
        L[(k+2)**n+k]=l[k+2][0];
    }
  }
}


void vorwaerts(double *L,
               int *n,
               double *QTY,
               double *CCC)
{
  int i,k,j;
  if (L[0]==0.0)
    printf("Nicht lösbar! L[0]=0.0\n");
  else{
    CCC[0]=QTY[0]/L[0];
    for(k=1;k<*n;k++)
      if (L[k**n+k]==0.0){
        printf("Nicht lösbar! L[%i][%i]=0.0\n",k,k);
        break;
      }
      else{
        CCC[k]=0.0;
        for(i=0;i<=(k-1);i++)
          CCC[k]+=L[k**n+i]*CCC[i];
        CCC[k]=1/L[k**n+k]*(QTY[k]-CCC[k]);
      }
  }
}


void rueckwaerts(double *L,
                 int *n,
                 double *GGG,
                 double *CCC)
{
  int i,k,l;
  if (L[(*n-1)**n+(*n-1)]==0.0)
    printf("Nicht lösbar! L[%i]=0.0\n",(*n-1)**n+(*n-1));
  else{
    GGG[*n-1]=CCC[*n-1]/L[(*n-1)**n+(*n-1)];
    for(k=*n-2;k>=0;k=k-1)
      if (L[k**n+k]==0.0){
        printf("Nicht lösbar! L[%i][%i]=0.0\n",k,k);
        break;
      }
      else{
        GGG[k]=0.0;
        if(k-1<0) l=0;
        else l=k-1;
        for(i=l;i<=(*n-1);i++)
          GGG[k]+=L[i**n+k]*GGG[i];
        GGG[k]=1/L[k**n+k]*(CCC[k]-GGG[k]);
      }
  }
}


void neben(double *Q,
           double *w,
```

```
          double *a,
          double *R,
          int *n,
          double *erg)
{
  int i,j,k,l,r,ll,rr;
  for (k=0;k<(*n-2);k++){
    if((k+2)>(*n-3)) r=*n-3;
    else r=k+2;
    for(j=k;j<=r;j++){
      erg[k*(*n-2)+j]=0.0;
      if(k>j) ll=k;
      else ll=j;
      if(*n>(j+2)) rr=*n;
      else
        if(*n>(k+2)) rr=*n;
        else
          if((j+2)>(k+2)) rr=j+2;
          else rr=k+2;
      for(i=ll;i<rr;i++)
        erg[k*(*n-2)+j]+=w[i]*Q[k*(*n)+i]*Q[j*(*n)+i];
      erg[k*(*n-2)+j]*=*a;
      erg[k*(*n-2)+j]+=R[j*(*n-2)+k];
      if (j>k)
        erg[j*(*n-2)+k]=erg[k*(*n-2)+j];
    }
  }
}


void SPLlokanp(double *x,
               double *y,
               int *n,
               double *w,
               double *thresh,
               double *reg,
               int *glob,
               int *mr,
               int *nit)
{
  int i,j,k;
  double h[*n-1], *qty, *Q, *R, *T, *w1,*L, *D, *G, *res;
  double *a;
  int *Z, STP, STT, *F, *N, counter,r;
  a=(double *)malloc(1*sizeof(double));
  Z=(int *)malloc(1*sizeof(int));
  F=(int *)malloc(1*sizeof(int));
  N=(int *)malloc(1*sizeof(int));
  qty = (double *)malloc((*n-2)*sizeof(double));
  Q = (double *)calloc(*n*(*n-2),sizeof(double));
  R = (double *)calloc((*n-2)*(*n-2),sizeof(double));
  T = (double *)calloc((*n-2)*(*n-2),sizeof(double));
  L = (double *)calloc((*n-2)*(*n-2),sizeof(double));
  w1 = (double *)malloc(*n*sizeof(double));
  D = (double *)malloc((*n-2)*sizeof(double));
  G = (double *)malloc((*n-2)*sizeof(double));
  res= (double *)malloc(*n*sizeof(double));
  counter=0;
  *a = 0.01;
  *Z = 1;
  STP=1;
  for(i=0;i<(*n-1);i++)
    h[i] = x[i+1]-x[i];
  for(i=0;i<(*n-2);i++)
    qty[i] = (y[i+2]-y[i+1])/h[i+1]-(y[i+1]-y[i])/h[i];
  for(i=0;i<(*n-2);i++){
    Q[i**n+i] = 1.0/h[i];
    Q[i**n+i+1] = -1.0/h[i]-1.0/h[i+1];
    Q[i**n+i+2] = 1.0/h[i+1];
    if(i<(*n-3)){
      R[i*(*n-2)+i] = 1.0/3.0*(h[i]+h[i+1]);
      R[i*(*n-2)+i+1] = R[(i+1)*(*n-2)+i] = 1.0/6.0*h[i+1];
    }
  }
  R[(*n-3)*(*n-2)+(*n-3)]=1.0/3.0*(h[*n-3]+h[*n-2]);
  *N = *n-2;
  if(*mr==1)
    do{
      counter+=1;
      printf("counter: %i\n", counter);
      *F=0;
      STP=0;
      for(i=0;i<*n;i++)
        w1[i] = 1.0/w[i];
      neben(Q,w1,a,R,n,T);
      cholesky(T, N, L, F);
      if(*F==1) break;
      vorwaerts(L,N, qty,D);
```

```
    rueckwaerts(L, N, G, D);
    for(j=2;j<(*n-2);j++){
      reg[j] = 0.0;
      for(i=j-2;i<=j;i++){
        reg[j]+=Q[i**n+j]*G[i];
      }
      reg[j] = y[j]-w1[j]**a*reg[j];
    }
    reg[0] = y[0]-w1[0]**a*Q[0]*G[0];
    reg[1] = y[1]-w1[1]**a*(Q[1]*G[0]+Q[1**n+1]*G[1]);
    reg[*n-2] = y[*n-2]-w1[*n-2]**a*(Q[(*n-4)**n+
                *n-2]*G[*n-4]+Q[(*n-3)**n+*n-2]*G[*n-3]);
    reg[*n-1] = y[*n-1]-w1[*n-1]**a*Q[(*n-3)**n+*n-1]*G[*n-3];
    for(i=0;i<*n;i++) res[i] = y[i]-reg[i];
    multiwdwr(res,n,thresh, Z);
    STT=0;
    if(*glob>0)
      for(i=0;i<*n;i++){
        if(res[i]==1.0){
          for(k=0;k<*n;k++)
            w[k]*=2.0;
          STP=1;
          STT=1;
        }
        if(STT==1)
          break;
      }
    else
      for(i=0;i<*n;i++){
        if(res[i]==1.0){
          w[i]*=2.0;
          STP=1;
        }
      }
  }while (STP==1);
else
  do{
    counter+=1;
    printf("counter: %i\n", counter);
    *F=0;
    STP=0;
    for(i=0;i<*n;i++)
      w1[i] = 1.0/w[i];
    neben(Q,w1,a,R,n,T);
    cholesky(T, N, L, F);
    if(*F==1) break;
    vorwaerts(L,N, qty,D);
    rueckwaerts(L, N, G, D);
    for(j=2;j<(*n-2);j++){
      reg[j] = 0.0;
      for(i=j-2;i<=j;i++){
        reg[j]+=Q[i**n+j]*G[i];
      }
      reg[j] = y[j]-w1[j]**a*reg[j];
    }
    reg[0] = y[0]-w1[0]**a*Q[0]*G[0];
    reg[1] = y[1]-w1[1]**a*(Q[1]*G[0]+Q[1**n+1]*G[1]);
    reg[*n-2] = y[*n-2]-w1[*n-2]**a*(Q[(*n-4)**n+
                  *n-2]*G[*n-4]+Q[(*n-3)**n+*n-2]*G[*n-3]);
    reg[*n-1] = y[*n-1]-w1[*n-1]**a*Q[(*n-3)**n+*n-1]*G[*n-3];
    for(i=0;i<*n;i++) res[i] = y[i]-reg[i];
    multiwdwr(res,n,thresh, Z);
    STT=0;
    if(*glob>0)
      for(i=0;i<*n;i++){
        if(res[i]==1.0){
          for(k=0;k<*n;k++)
            w[k]*=2.0;
          STP=1;
          STT=1;
        }
        if(STT==1)
          break;
      }
    else
      for(i=0;i<*n;i++){
        if(res[i]==1.0){
          w[i]*=2.0;
          STP=1;
        }
      }
  }while(counter<*nit);
free(N);
free(a);
free(Z);
free(F);
free(qty);
free(Q);
```

```
  free(R);
  free(T);
  free(w1);
  free(L);
  free(D);
  free(G);
  free(res);
}
```

Bibliography

[1] F. Abramovich and D. M. Steinberg. Improved inference in nonparametric regression using l_k-smoothing splines. *Journal of Statistical Planning and Inference*, 49(49):327–341, 1996.

[2] M. Brockmann, T. Gasser, and E. Herrmann. Locally adaptive bandwidth choice for kernel regression estimators. *J. Am. Stat. Ass.*, 88:1302–1309, 1993.

[3] A. Chambolle and P.-L. Lions. Image recovery via total variation minimization and related problems. *Numer. Math.*, 76(2):167–188, 1997.

[4] M. Chambolle. An algorithm for total variation minimization and applications. *Journal of Mathematical Imaging and Vision*, 20:89–97, 2004.

[5] C.-K. Chu and J. S. Marron. Choosing a kernel regression estimator. *Statist. Sci.*, 6(4):404–436, 1991. With comments and a rejoinder by the authors.

[6] W. S. Cleveland. Robust locally weighted regression and smoothing scatterplots. *J. Amer. Statist. Assoc.*, 74(368):829–836, 1979.

[7] D.D. Cox. Asymptotics for M-type smoothing splines. *Ann. Statist.*, 11(2):530–551, 1983.

[8] P. Craven and G. Wahba. Smoothing noisy data with spline functions. Estimating the correct degree of smoothing by the method of generalized cross-validation. *Numer. Math.*, 31(4):377–403, 1978/79.

[9] P.L. Davies and U. Gather. The identification of multiple outliers. *J. Amer. Statist. Assoc.*, 88(423):782–801, 1993. With discussion and a reply by the authors.

[10] P.L. Davies and A. Kovac. Local extremes, runs, strings and multiresolution. *The Annals of Statistics*, 29:1–65, 2001.

[11] C. de Boor. *A practical guide to splines*, volume 27 of *Applied Mathematical Sciences*. Springer-Verlag, New York, 1978.

[12] C. de Boor. Calculation of the smoothing spline with weighted roughness measure. *Mathematical Models and Methods in Applied Sciences*, 11:33–41, 2001.

[13] D.L. Donoho. Nonlinear solution of linear inverse problems by wavelet-vaguelette decomposition. *Appl. Comput. Harm. Anal.*, 2:101–126, 1995.

[14] D.L. Donoho and I.M. Jonstone. Ideal spatial adaptation by wavelet shrinkage. *Biometrika*, 81:425–455, 1994.

[15] J. Duchon. Fonction-"spline" associée à n observations d'une fonction aléatoire. *C. R. Acad. Sci. Paris Sér. A-B*, 280:Aii, A949–A951, 1975.

[16] J. Duchon. Fonctions-spline et espérances conditionnelles de champs gaussiens. *Ann. Sci. Univ. Clermont*, (61 Math. No. 14):19–27, 1976.

[17] J. Duchon. Splines minimizing rotation-invariant semi-norms in Sobolev spaces. pages 85–100. Lecture Notes in Math., Vol. 571, 1977.

[18] L. Dümbgen. Confidence bands for convex median curves using sign-tests. 2001.

[19] L. Dümbgen and R.B. Johns. Confidence bands for isotonic median curves using sign tests. 2000.

[20] R.L. Eubank. *Spline Smoothing and Nonparametric Regression.* Marcel Dekker, New York, 1988.

[21] R.L. Eubank. *Nonparametric Regression and Spline Smoothing.* Marcel Dekker, New York, 1999.

[22] J. Fan. Design-adaptive nonparametric regression. *Journal of the American Statistical Association*, 87:998–1004, 1992.

[23] J. Fan. Local linear regression smoothers and their minimax efficiencies. *The Annals of Statistics*, 21:196–216, 1993.

[24] J. Fan and I. Gijbels. Variable bandwidth and local linear regression smoothers. *Ann. Statist.*, 20(4):2008–2036, 1992.

[25] J. Fan and I. Gijbels. *Local Polynomial Modelling and Its Applications.* Chapman and Hall, London, 1996.

[26] T. Gasser, A. Kneip, and W. Köhler. A flexible and fast method for automatic smoothing. *Journal of the American Statistical Association*, 86:643–652, 1991.

[27] T. Gasser and H.-G. Müller. Kernel estimation of regression functions. *Smoothing Techniques for Curve Estimation, Lecture Notes in Mathematics*, 757:23–68, 1979.

[28] T. Gasser and H.-G. Müller. Estimating regression functions and their derivatives by the kernel method. *Scandinavian Journal of Statistics*, 11:171–185, 1984.

[29] P.J. Green and B.W. Silverman. *Nonparametric Regression and Generalized Linear Models.* Chapman & Hall, London e.a., 1994.

[30] M. Hansen, C. Kooperberg, and Sylvain Sardy. Triogram models. *Journal of the American Statistical Association*, 93(441):101–119, 1998.

[31] W. Härdle, P. Hall, and J. S. Marron. How far are automatically chosen regression smoothing parameters from their optimum? *J. Amer. Statist. Assoc.*, 83(401):86–101, 1988. With comments by David W. Scott and Iain Johnstone and a reply by the authors.

[32] W. Härdle, P. Hall, and J. S. Marron. Regression smoothing parameters that are not far from their optimum. *J. Amer. Statist. Assoc.*, 87(417):227–233, 1992.

[33] W. Härdle and J. S. Marron. Optimal bandwidth selection in nonparametric regression function estimation. *Ann. Statist.*, 13(4):1465–1481, 1985.

[34] T. J. Hastie and R. J. Tibshirani. *Generalized additive models*, volume 43 of *Monographs on Statistics and Applied Probability*. Chapman and Hall Ltd., London, 1990.

[35] T.J. Hastie and C. Loader. Local regression: automatic kernel carpentry (with discussion). *Statist. Sci.*, 8:120–143, 1993.

[36] E. Herrmann. Local bandwidth choice in kernel regression estimation. *J. Am. Stat. Ass.*, 6:35–54, 1997.

[37] W. Härdle. *Applied nonparametric regression*. Cambridge University Press, New York u.a., 1990.

[38] R. Ihaka and R. Gentlemen. R: A language for data analysis and graphics. *Journal of Computational and Graphical Statistics*, 5:299–314, 1996.

[39] R. Koenker and I. Mizera. Penalized triograms: total variation regularization for bivariate smoothing. *J. R. Stat. Soc. Ser. B Stat. Methodol.*, 66(1):145–163, 2004.

[40] R. Koenker and P. Ng. Sparsem: a sparse matrix package for r. *J. Statist. Softwr.*, 8(6):1–9, 2002.

[41] R. Koenker, P. Ng, and S. Portnoy. Quantile smoothing splines. *Biometrika*, 81(4):673–680, 1994.

[42] R. Koenker and S. Portnoy. The Gaussian hare and the Laplacian tortoise: computability of squared-error versus absolute-error estimators. *Statist. Sci.*, 12(4):279–300, 1997.

[43] A. Kovac. Robust nonparametric regression and modality. In *Developments in robust statistics (Vorau, 2001)*, pages 218–227. Physica, Heidelberg, 2003.

[44] A. Majidi. *Glatte nichtparametrische Regression unter formerhaltenden Bedingungen*. PhD thesis, Universität Essen, 2003.

[45] J. Meinguet. Multivariate interpolation at arbitrary points made simple. *Z. Angew. Math. Phys.*, 30(2):292–304, 1979.

[46] M. Meyer and M. Woodroofe. On the degrees of freedom in shape-restricted regression. *Ann. Statist.*, 28(4):1083–1104, 2000.

[47] H.-G. Müller. Smooth optimum kernel estimators near endpoints. *Biometrika*, 78:521–530, 1991.

[48] H.-G. Müller and U. Stadtmüller. Variable bandwidth kernel estimators of regression curve. *The Annals of Statistics*, 15:182–201, 1987.

[49] E.A. Nadaraya. On estimating regression. *Theory Prob. Appl.*, 10:186–190, 1964.

[50] Jorge Nocedal and Stephen J. Wright. *Numerical optimization*. Springer Series in Operations Research. Springer-Verlag, New York, 1999.

[51] J. Polzehl and V. Spokoiny. Image denoising: pointwise adaptive approach. *Ann. Statist.*, 31(1):30–57, 2003.

[52] J. Polzehl and V.G. Spokoiny. Adaptive weights smoothing with applications to image restoration. *J. R. Stat. Soc. B*, 62:335–354, 2000.

[53] M. B. Priestley and M. T. Chao. Non-parametric function fitting. *J. Roy. Statist. Soc. Ser. B*, 34:385–392, 1972.

[54] C. Reinsch. Smoothing by spline functions. *Numerische Mathematik*, 10:177–183, 1967.

[55] M. Rosenblatt. *Stochastic Curve Estimation*. Institute of Mathematical Statistics, Hayward, CA, 1991.

[56] L. I. Rudin, S. Osher, and E. Fatemi. Nonlinear total variation based noise removal algorithms. *Physica D*, 60:259–268, 1992.

[57] D. Ruppert and M.P. Wand. Multivariate locally weighted least squares regression. *The Annals of Statistics*, 22:1346–1370, 1994.

[58] L. L. Schumaker. *Spline functions: basic theory.* John Wiley & Sons Inc., New York, 1981.

[59] B. W. Silverman. Spline smoothing: the equivalent variable kernel method. *Ann. Statist.*, 12(3):898–916, 1984.

[60] B. W. Silverman. Some aspects of the spline smoothing approach to nonparametric regression curve fitting. *J. Roy. Statist. Soc. Ser. B*, 47(1):1–52, 1985. With discussion.

[61] J.G. Staniswalis. Local bandwidth selection for kernel estimates. *J. Am. Stat. Ass.*, 84:284–288, 1989.

[62] C.J. Stone. Consistent nonparametric regression. *Ann. Statist.*, 5:595–620, 1977.

[63] F. Utreras. Natural spline functions, their associated eigenvalue problem. *Numer. Math.*, 42(1):107–117, 1983.

[64] G. Wahba. A survey of some smoothing problems and the method of generalized cross-validation for solving them. *Applications of Statistics*, pages 507–523, 1977.

[65] G. Wahba. *Spline models for observational data*, volume 59 of *CBMS-NSF Regional Conference Series in Applied Mathematics*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1990.

[66] M.P. Wand and M.C. Jones. *Kernel Smoothing.* Chapman & Hall, London e.a., 1995.

[67] G.S. Watson. Smooth regression analysis. *Sankhyā Ser. A*, 26:359–372, 1964.

[68] S. J. Wright. *Primal-dual interior-point methods.* Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1997.